Design and Evaluation of a Recommender System

INF-3981 Master's Thesis in Computer Science

Magnus Mortensen

Faculty of Science
Department of Computer Science
University of Tromsø

February 5, 2007

Design and Evaluation of a Recommender System

INF-3981 Master's Thesis in Computer Science

Magnus Mortensen

Faculty of Science
Department of Computer Science
University of Tromsø

February 5, 2007

Abstract

In the recent years, the Web has undergone a tremendous growth regarding both content and users. This has lead to an information overload problem in which people are finding it increasingly difficult to locate the right information at the right time.

Recommender systems have been developed to address this problem, by guiding users through the big ocean of information. Until now, recommender systems have been extensively used within e-commerce and communities where items like movies, music and articles are recommended. More recently, recommender systems have been deployed in online music players, recommending music that the users probably will like.

This thesis will present the design, implementation, testing and evaluation of a recommender system within the music domain, where three different approaches for producing recommendations are utilized.

Testing each approach is done by first conducting live user experiments and then measure recommender precision using offline analysis. Our results show that the functionality of the recommender system is satisfactory, and that recommender precision differs for the three filtering approaches.

Acknowledgments

The author would like to thank his supervisor, Professor Dag Johansen, for valuable ideas, support and motivation.

Many thanks to the test users. The experiment would not be possible without your help.

Thanks to Åge Kvalnes for server support.

Thanks to the rest of the WAIF team for providing valuable input.

Also thanks to FAST Search & Transfer, in particular represented by Krister Mikalsen, for helpful discussions.

Finally, thanks to girlfriend Ragnhild Høifødt for reading through the draft and providing support.

Contents

Chapter 1

Introduction

1.1 Background

The World Wide Web contains an enormous amount of information. In January 2005 the number of pages in the publicly indexable [54] web was exceeding 11.5 billion [26]. Recent statistics also show that the number of Internet users is high and rapidly growing. Statistics from September 18th 2006 shows that 17% of the world's population uses the Internet and that the number of users has grown with over 200% from 2000 to 2006 [1].

The tremendous growth of both information and usage has lead to a so-called information overload problem in which users are finding it increasingly difficult to locate the right information at the right time [48]. As a response to this problem, much research has been done with the goal of providing users with more proactive and personalized information services.

Recommender systems have proved to help achieving this goal by using the opinions of a community of users to help individuals in the community more effectively identify content of interest from a potentially overwhelming set of choices [49]. Two recommendation strategies that have come to dominate are content-based and collaborative filtering. Content-based filtering rely on rich content descriptions of the items that are being recommended [43], while collaborative filtering recommendations are motivated by the observation that we often look to our friends for recommendations [52].

Systems using recommendations have been developed in various research projects. The system called Tapestry [25] is often associated with the genesis of computer-based recommendation systems. Later, several research projects have focused on recommender systems, either by introducing new concepts, or by combining old concepts to make better systems.

Recommender systems have also been deployed within commercial domains, for example in e-commerce applications. A well-known example is Amazon[1], where a recommender

[1] www.amazon.com

system is used to help people find items they would like to purchase. Many online communities within the movie domain use recommender systems to gather user opinions on movies, and then produce recommendations based on these opinions. Examples are MovieFinder[2] and Movielens[3]. New popular music services like Pandora[4] and Last.fm[5] also make use of recommendations to configure personalized music players.

1.2 Problem definition

This thesis shall focus on development and evaluation of a recommender system within the music domain. Different approaches for computing recommendations will be designed, implemented and tested with real end-users. Evaluation will be done by assessing the system functionality and comparing the recommender precision obtained by each approach.

1.3 Interpretation

Throughout the last years, recommender systems have been deployed in various personalized music players. One reason for the success behind these players is due to their ability to produce recommendations that accurately suits their users. By developing and testing different variants of a music player using standard recommendation strategies, we might be able to discover how the different techniques influence recommender precision.

We also conjecture that the standard strategies are not always sufficient to reflect a person's preference, where preference often is context dependent [4]. One important aspect of a person's context is mood. By integrating a mechanism for mood filtering into the music recommender system, it may be possible to give recommendations that better suits a person's often varying music preference.

Three variants of the recommender system will be tested using content-based filtering, collaborative filtering and contextual collaborative filtering respectively. Testing includes user experiments, where the users evaluate and listen to recommended music while the system receives user feedback. Since listening to a vast variety of music generally takes time, we conjecture that the users normally will test the system during the week while studying or working. After testing the system, user feedback will be used to calculate recommender precision. Finally, the results will be presented and evaluated.

1.4 Method and approach

The discipline of computing is divided into three paradigms[19]. These are *theory*, *abstraction* and *design*.

Theory is based on mathematics, and consists of the following four steps.

1. Characterize the objects under study.

[2] www.moviefinder.com
[3] www.movielens.umn.edu
[4] www.pandora.com
[5] www.last.fm

2. Make hypothesis about the relationship between the objects.
3. Prove hypothesis true or false.
4. Interpret the result.

Abstraction is based on experimental scientific methods, and consists of the following steps for investigating a phenomenon.

1. Form a hypothesis.
2. Construct a model.
3. Design an experiment and collect data.
4. Analyze the result.

Design is a paradigm rooted in engineering, and consists of four steps for constructing a system that shall solve a problem.

1. Sate requirements and specifications for the system.
2. Design the system.
3. Implement the system.
4. Test the system.
5. Evaluate the system.

This thesis will follow the design paradigm, which means that a system solving a specific problem will be developed. The problem is reflected in the requirement specification. To fulfill these requirements, the system will be designed and implemented. Testing will be done to measure system performance, in our case functionality and recommender precision. Finally, the system will be evaluated by consulting test results and requirements, and then consider alternative approaches.

1.5 Outline

This thesis consists of the following chapters:

Chapter 2 - Related Work introduces the information overload problem that current web technologies have to deal with. Different solutions to the problem is presented, focusing on recommender systems. The chapter ends with a case study, comparing two recommender systems that are relevant for our work.

Chapter 3 - Requirements states the system requirements.

Chapter 4 - Design proposes the design of our music recommender system and its components.

Chapter 5 - Implementation discusses technical considerations and describes the implementation of the system.

Chapter 6 - Experiment describes the experiments carried out for this thesis, and the experimental results.

Chapter 6 - Evaluation evaluates the system with respect to the requirements.

Chapter 7 - Conclusion draws the conclusion of this thesis and recommends possible future work.

Chapter 2

Related work

This chapter will introduce the problem that recommender systems are trying to solve, and different approaches for solving this problem. Present techniques used to improve recommender systems are also explained before describing a case study with a comparison between two popular recommender systems.

2.1 The World Wide Web

The World Wide Web (WWW or web) emerged in the early nineties. In 1990 Tim Berners-Lee emphasized the necessity of an information management system to prevent the loss of information resulting from the growing organizational structure at CERN[1] [8].

The technology behind the web can be characterized as an information system composed of agents [2]. Agents are programs that act on behalf of a person, entity or process to exchange or process information. The main types of agents are server agents and client agents. A server agent offers services that are used by the client agents, as shown in figure 2.1. When a user follows a link on a web page in the browser, the browser performs a request to the server, which responds by returning a web page.

The World Wide Web consortium was founded in October 1994. The goal was to develop and maintain protocols that specify the standards that enable computers on the web to

[1]The European Organization for Nuclear Research

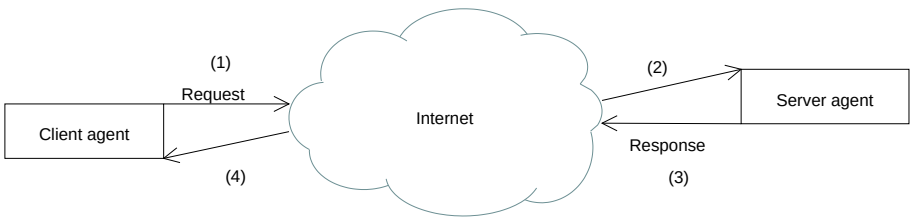

Figure 2.1: Client and server interaction.

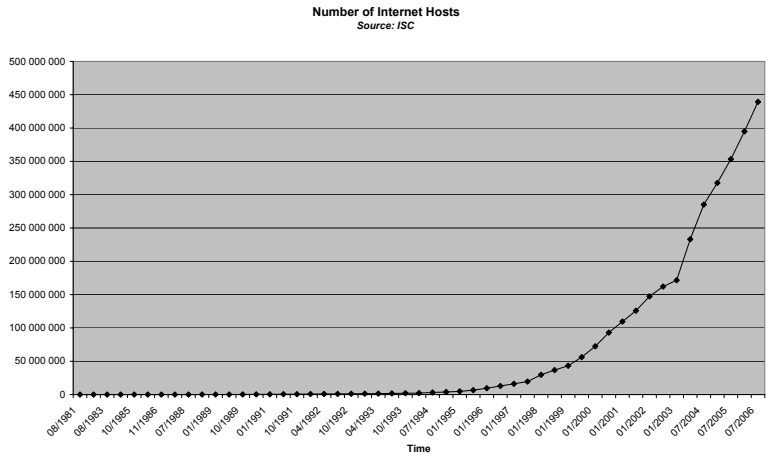

Figure 2.2: The number of Internet hosts between 1981 and 2006.

effectively store and communicate different forms of information. At its core, the web is made up of the following standards:

Hypertext transfer protocol (HTTP) [9] is a transport protocol that specifies how the servers and clients communicate with each other. When a user types the URL of a web page or follows a link on a web page, the user's web browser performs a HTTP request to the server. The server responds by returning the web page content in quick successions.

Hypertext mark-up language (HTML) [17] is used to define the structure of web pages. The language has notions for embedding references to other documents. These references appear on web pages as hyperlinks that the users can select to fetch and display the referenced page. Recently, another markup language, XML [12], has been defined to facilitate the sharing of data across different information systems.

Uniform resource locator (URL[2]*)* [11] is a universal system for referencing resources on the web.

Together, these standards form a simple and effective platform for sharing information. Due to this, and the fact that computers and Internet access have become more available, the World Wide Web has undergone an exponential growth, both in number of computers and users. Figure 2.2 shows the growth in the number of Internet hosts between 1981 and 2006 [57].

As the World Wide Web continues to grow at an exponential rate, the size and complexity of web pages grow along with it. Different techniques have been applied to develop systems that help users find the information they seek. These techniques belong to the fields in software technology called *information retrieval* and *information filtering*.

[2]URI has been defined in [10]

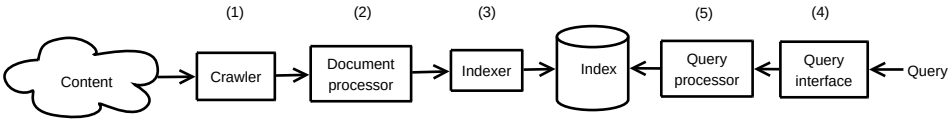

Figure 2.3: Information retrieval.

2.2 Information retrieval and filtering

The rapidly expanding Internet has given the users the ability to choose among a vast variety of information [27], whether it is information concerning their profession, events in their world, or information that allows them to maintain their lifestyle. The information that is needed to fulfil these continuously increasing demands can come from different sources. Examples are web pages, emails, articles, news, consumer journals, shopping sites, online auctions and multimedia sites. Even though the users profit from the enormous amount of information that the sources provide, they are not able to handle it. This information overload problem [48] is the reason why several techniques for *information retrieval* and *information filtering* have been developed. Although the goal of both information retrieval and information filtering is to deal with the information overload problem by examining and filtering big amounts of data, there is often made a distinction between the two [7].

Information retrieval

Information retrieval (IR), often associated with data search, is a technology that may include *crawling*, *processing* and *indexing* of content, and *querying* for content. The normal process of IR is showed in figure 2.3. Crawling is the act of accessing web servers and/or file systems in order to fetch information. By following links, a crawler is able to traverse web content hierarchies based on a single start URL. The document-processing stage may add, delete or modify information to a document, such as adding new meta information for linguistic processing, or extracting information about the language that the document is written in. Indexing is a process that examines content that has been processed and makes a searchable data structure, called Index, that contains references to the content.

Queries are requests for information. IR systems let a user write a query in form of keywords describing the information needed. The user can interact with the IR system through a Query interface. A Query-processor will use the index to find information references based on the keywords and then display the references. The goal is to analyze and identify the essence of the user's intent from the query, and to return the most relevant set of results.

Filtering of information in IR systems is done by letting the user specify what information is needed by manually typing keywords describing the information. IR is very successful at supporting users who know how to describe exactly what they are looking

for in a manner that is compatible with the descriptions of the content that were created during the indexing.

Information filtering

Information filtering (IF) systems focus on filtering information based on a user's *profile*. The profile can be maintained by letting the user specify and combine interests explicitly, or by letting the system implicitly monitor the user's behavior.

Filtering within IF systems is done when the user automatically receives the information needed based on the user's profile. The advantage of IF is its ability to adapt to the user's long-term interest, and bring the information to the user. The latter can be done by giving a notice to the user, or by letting the system use the information to take action on behalf of the user.

Closely related to IF is the idea of having a system that acts as a personalized decision guide for users, aiding them in decision making about matters related to personal taste. Systems that realize this idea are called *recommender systems*.

2.3 Recommender systems

"We have 6.2 million customers; we should have 6.2 million stores. There should be the optimum store for each and every customer."

—Jeff Bezos, CEO of Amazon.com™[3]

In everyday life, when presented with a number of unfamiliar alternatives, people normally tend to ask friends for guidance, or to seek expert help by reading reviews in magazines and newspapers. In the recent years, online *recommender systems* have begun to provide a technological proxy for this social recommendation process [58], in which they are used to either predict whether a particular user will like a particular item *(prediction)*, or to identify a set of N items that will be of interest to a certain user *(top-N recommendation)*.

Recommender systems (RS) [49] are used in a variety of applications. Examples are web stores, online communities, and music players. Currently, people mostly tend to associate recommender systems with e-commerce sites, where recommender systems are extensively used to suggest products to the customers and to provide customers with information to help them decide which products to purchase. Products can be based on the top overall sellers on a site, on the demographics of the consumers, or on an analysis of the past buying behaviour of the consumers as a prediction for future buying behaviour [53]. This is shown in figure 2.4.

Content-based filtering and *collaborative filtering* are two algorithmic techniques for computing recommendations. A content-based filtering system selects items based on the correlation between the content and the user's preference, as opposed to a collaborative filtering system that chooses items based on the correlation between people with similar

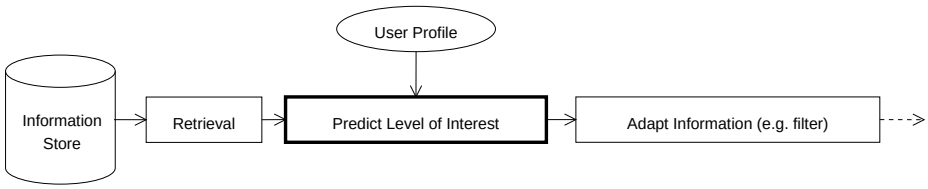

Figure 2.4: Information filtering in recommender systems.

preference. Systems using the latter technique are also referred to as *automatic recommender systems* [53]. In addition, a *hybrid* approach has been developed to avoid certain limitations related to content-based and collaborative filtering.

2.3.1 Content-based filtering

Because of the information overload problem explained in chapter 2.2, researchers have been working for more than thirty years with technologies that allow automatic categorization and recommendation of information to a user based on the user's personal preferences [27]. In particular, various candidate items are compared with items previously rated by the user and the best-matching items are recommended.

Information filtering differs from information retrieval in the way the interests of a user are presented. Instead of letting the user *pull* information using a query, an information filtering system tries to model the user's long term interests and *push* relevant information to the user. Despite this difference, information filtering have borrowed certain techniques from information retrieval [27], as is reflected in content-based filtering, and also in collaborative filtering. One technique is *term frequency indexing* [50], where documents and user preferences are represented by vectors. As figure 2.5 shows, the vector space have one dimension for each word in the database. Each part of the vector is the frequency that the respective word occurs in the document or the user query. The document vectors that are found to be the closest to the query vectors are possibly most relevant to the user's query. Collaborative filtering systems can use this technique by letting each user profile be represented by a vector, and then compare user similarities by interpreting the vectors.

Other techniques borrowed from IR systems include Boolean search indexes, where keywords in a query are combined with Boolean operators [27, 16]; probabilistic retrieval systems where probabilistic reasoning is used to determine the probability that a document meets a user's information need [23]; and natural language query interfaces, where queries are posed in natural sentences [39].

There are several examples of systems that use content-based filtering to assist users in finding information [60]. Letizia [40] is a user interface that assists users browsing the web. The system tracks the browsing behaviour of a user and tries to anticipate what pages may be of interest to the user. Syskill & Webert [45] is a system that based on a user's rating of web pages over time predict which web pages will interest the user.

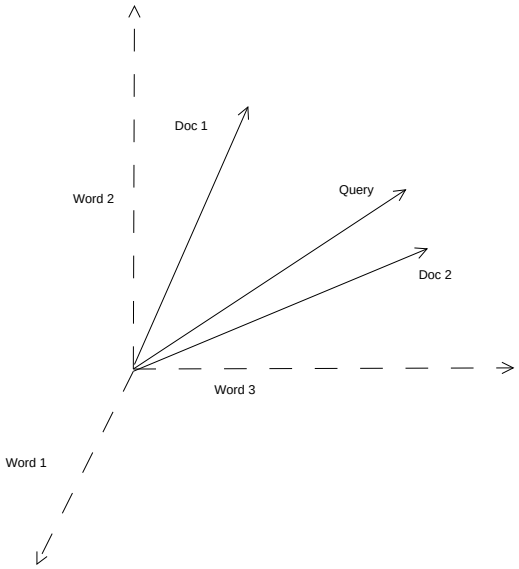

Figure 2.5: Term frequency indexing.

Higuchi [34] is a system using a neural network to model a user's interests within a news environment. The neural network is maintained while the user is approving or rejecting different articles. What these systems have in common is that they all operate on textual information.

Information only consisting of text can easily be parsed with today's technology, and then automatically categorized. For other types of information like multimedia data (e.g. images, music and movies), the categorization requires more complex operations. The technology for parsing multimedia data is getting better, but it will still take a while before it can be done without human interaction. Today, categorization of such information is mostly done manually by humans. This activity is expensive, time-consuming, error-prone and highly subjective [42]. For this reason, content-based systems are not suitable for dynamic, large environments with a vast and variant amount of information. However, if information can be categorized without having to parse the information, this problem can be avoided.

2.3.2 Collaborative filtering

The difficulties of automatic information processing have put restrictions on content-based filtering technology. Collaborative filtering (CF) [25] was developed to address this weakness. CF is different from other filtering technologies in that information is filtered by using evaluation instead of analysis, thus categorizing information based on the user's opinion of the information instead of the information itself. In addition, CF stresses the concept of community by letting recommendations be a result of the opinions of the current user and other similar users. As figure 2.6 shows, all users contribute with ratings based on their preferences. Recommendations for the current user are produced by matching the user's ratings with ratings given by other users. In this way, similar

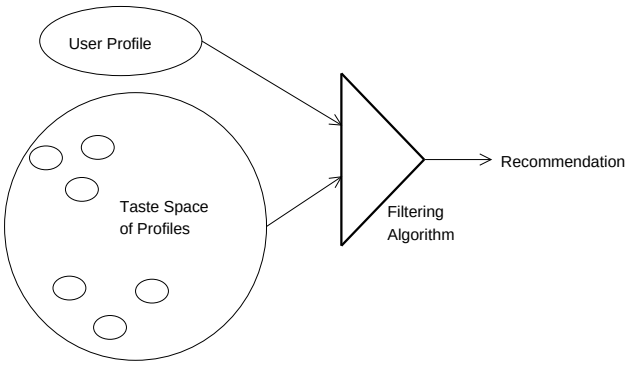

Figure 2.6: Collaborative Filtering.

users are linked together to form a community.

The properties of CF make it possible to build systems that have advantages above what is possible with content-based filtering. First, because recommendations are independent of the content itself, it is possible to filter information from any source. Second, it is possible to filter and recommend information based on deep and complex relationships to the user, such as taste or quality. For example, CF makes it possible to differ between well-written and poorly written documents. Third, it is possible to receive serendipitous recommendations. These are recommendations for information that the user is not actively looking for. Imagine a music recommender system where a user have listened to several bad jazz songs, and conclude that jazz is not interesting. The user specifies in the recommender system that jazz is not of interest, and will then stop receiving jazz recommendations. However, assume that a second user, who dislikes the same jazz songs as the first user, finds a good jazz song. Then, CF will make sure that the jazz song is recommended to the first user. The user may then discover that jazz is not that bad after all. Finally, CF helps to create communities, as explained above. None of this would be possible using content-based filtering.

Systems using CF have been widely used in entertainment domains. Despite the fact that the technology is mostly accurate, it has yet to be successful in domains where a higher risk is associated with the acceptance of a recommendation. Users do not normally object to purchasing CDs or DVDs after receiving recommendations from systems using CF. However, a user would probably not take the risk of buying a house based on such recommendations. The fact that CF systems are not trusted for risky content domains has its explanation. Predictions made by recommender systems reflect approximations made by humans. They are therefore not always accurate, and certainly not objective. In addition, CF systems are doing calculations based on sparse and incomplete data. Together these two conditions explain why the recommendations given by CF systems are generally correct, but sometimes very wrong.

Another related issue concerns trust. CF systems act as black boxes, computerized oracles that give advice, but cannot be questioned [30]. The typical interaction paradigm

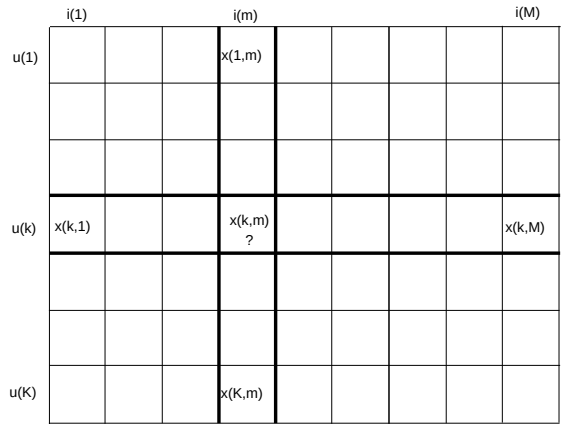

Figure 2.7: The user-item matrix.

involves asking the user for some input, processing the input, and giving the user some output in the form of recommendations. The user does not know why the specific content was recommended. This problem has also prevented acceptance in all but the low-risk content domains.

It is important to note that CF technologies normally do not compete with content-based filtering technologies. Today, the two technologies are usually integrated to provide powerful hybrid filtering solutions. Successful research has been done in projects like GroupLens [37, 48], Ringo [55], Video Recommander [31] and MovieLens [18]. Commercial sites using CF technology are Amazon[3], CDNow[4], MovieFinder[5] and Launch[6]. These research projects and commercial sites make use of different approaches to achieve collaborative filtering. These approaches will now be explained.

2.3.3 Collaborative filtering approaches

Collaborative Filtering systems are often classified as *memory-based(user-based)* or *model-based(item-based)*. Early research used a memory-based approach that makes rating predictions based on the entire collection of previously rated items by the users. Then, due to limitations with this approach, researchers developed model-based CF systems that use the collection of ratings to learn a *model*, which is used to make predictions. Although the model-based approach deals with some of the limitations related to memory-based CF, this approach also has its shortcomings. These approaches will be described further, and their strengths and weaknesses will be addressed.

User-item matrix

A user-item matrix can be used to describe memory-based and model-based CF [61]. For K users and M items, the user profiles are represented in a $K \times M$ user-item matrix $\mathbf{X}$, as

[3] www.amazon.com
[4] www.cdnow.com
[5] www.moviefinder.com
[6] www.launch.com

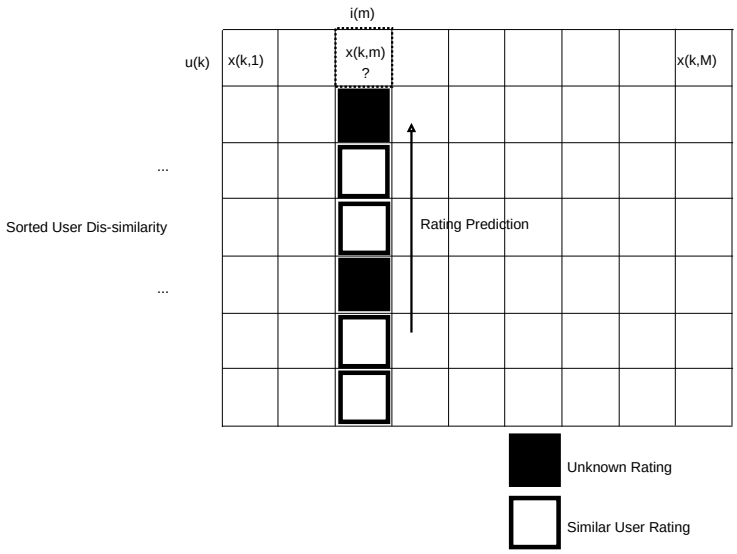

Figure 2.8: Rating prediction based on user similarity

in figure 2.7. Each element $x_{k,m} = r$ indicates that the user k rated item m by r, where $r \in \{1, ..., |r|\}$ if the item has been rated, and $x_{k,m} = \emptyset$ means that the rating is unknown.

The user-item matrix can be decomposed into row vectors:

$\mathbf{X} = [u_1, ..., u_K], u_k = [x_{k,1}, ..., x_{k,M}], k = 1, ..., K$

Each row vector u_k corresponds to a user profile and represents a particular user's item ratings. This decomposition leads to memory-based CF.

The matrix can also be represented by its column vectors:

$\mathbf{X} = [i_1, ..., i_M], i_m = [x_{1,m}, ..., i_{K,m}], m = 1, ..., M$

where each column vector i_m corresponds to a specific item's ratings by all K users. This representation shows model-based CF.

Memory-based collaborative filtering

A memory-based CF approach, or *nearest-neighbor* [13, 29, 35, 48] is said to form an implementation of the "Word of Mouth" phenomenon by maintaining a database of all the users known preferences of all items, and for each prediction perform some computation across the entire database. It predicts the user's interest in an item based on ratings of information from similar user profiles. This is shown in figure 2.8, where the prediction of a specific item (belonging to a specific user) is done by sorting the row vectors (user profiles) by its dissimilarity toward the user. In this way, ratings by more similar users will contribute more to the rating prediction.

A variety of memory-based CF systems have been developed [47]. The Tapestry system relied on each user to identify like-minded users manually [25]. GroupLens [48] and Ringo [55], developed independently, were the first CF algorithms to automate prediction.

These are general memory-based approaches, where for each prediction, computations are done over the entire database of user ratings. The Pearson correlation coefficient was used in GroupLens [48]. The Ringo project [55] focuses on testing different similarity metrics, including correlation and mean squared difference. Breese et al. [13] propose the use of vector similarity, based on the vector cosine measure often used in information retrieval systems. In addition to the research projects mentioned, a number of commercial systems using memory-based CF have been developed, most notably the systems deployed at Amazon and CDNow.

Memory-based CF methods have reached a high level of popularity because they are simple and intuitive on a conceptual level while avoiding the complications of a potentially expensive model-building stage. At the same time they are sufficient to solve many real-world problems. Yet there are some shortcomings [52, 32]:

> **Sparsity.** In practice, many memory-based CF systems are used to evaluate large sets of items. In these systems, even active users may have consumed well under 1% of the items. Accordingly, a memory-based CF system may be unable to make any item recommendation for a particular user. As a result, the recommendation accuracy can be poor.
>
> **Scalability.** The algorithms used by most memory-based CF systems require computations that grow according to the number of users and items. Because of this, a typical memory-based CF system with millions of users and items will suffer from serious scalability problems.
>
> **Learning.** Since no explicit statistical model is constructed, nothing is actually learned from the available user profile and no general insight is gained.

The weaknesses of memory-based CF systems, especially the scalability and learning issue have led to the exploration of an alternative model-based CF approach.

Model-based collaborative filtering

The motivation behind model-based CF is that by compiling a model that reflects user preferences, some of the problems related to memory-based CF might be solved. This can be done by first compiling the complete data set into a descriptive model of users, items and ratings. This model can be built off-line over several hours or days. Recommendations can then be computed by consulting the model.

Instead of using the similarity of users to predict the rating of an item, the model-based approach uses the similarity of items. This is illustrated in figure 2.9. Prediction is done by averaging the ratings of similar items rated by the specific user [20, 52, 41]. Sorting is done according to dissimilarity, as in memory-based CF. The difference is that the column vectors (items) are sorted toward the specific item, and not as in memory-based CF, where row vectors are sorted toward the specific user. Sorting of the column vectors assures that the ratings from more similar items are weighted stronger.

Early research on this approach evaluated two probabilistic models, *Bayesian clustering* and *Bayesian networks* [13]. In the Bayesian clustering model, users with similar preferences are clustered together into classes. Given the user's class membership, the ratings are assumed to be independent. The number of classes and the model parameters are learned from the data set. In the Bayesian network model, each node in the network corresponds to an item in the data set. The state of each node corresponds to the possible rating values for each item. Both the structure of the network, which encodes the dependencies between items, and the conditional probabilities, are learned from the data set.

Ungar and Foster [59] also suggest clustering as a natural pre-processing step for CF. Users and items are classified into groups. For each category of users, the probability that they like each category of items is estimated. Results of several statistical techniques for clustering and model estimation are compared, using both synthetic and real data.

Research have also focused on a rule-based approach for doing model-based CF. This approach applies association rule discovery algorithms to find associations between co-purchased items and then generates item recommendations based on the strength of the association between items [51].

As mentioned in section 2.3.3, memory-based CF approaches suffers from a data sparsity problem. Model-based methods solve this problem to a certain extent, due to their "compact" model. However, the need to tune a significant number of parameters has prevented these methods from practical usage. Lately, researchers have introduced dimensionality reduction techniques to address data sparsity [61], but as pointed out in [33, 62], some useful information may be discarded during the reduction. [33] has explored a graph-based method to deal with data sparsity, using transitive associations between users and items in the bipartite user item graph.

Another direction in collaborative filtering research combines memory-based and model-based approaches [47, 62]. [61] proposes a framework for including model-based recommendations into the final prediction, and does not require clustering of the data set a priori.

Model-based CF has several advantages over memory-based CF. First, the model-based approach may offer added values beyond its predictive capabilities, by highlighting certain correlations in the data. Second, memory requirements for the model are normally less than for storing the whole database. Third, predictions can be calculated quickly once the model is generated, though the time complexity to compile the data into a model may be prohibitive, and adding one new data point may require a full recompilation.

The resulting model of model-based CF systems is usually very small, fast and essentially as accurate as memory-based methods [13]. Model-based methods may prove practical for environments in which user preferences change slowly with respect to the time needed to build the model. Model-based methods, however, are not suitable for environments in which user preference models must be updated rapidly or frequently.

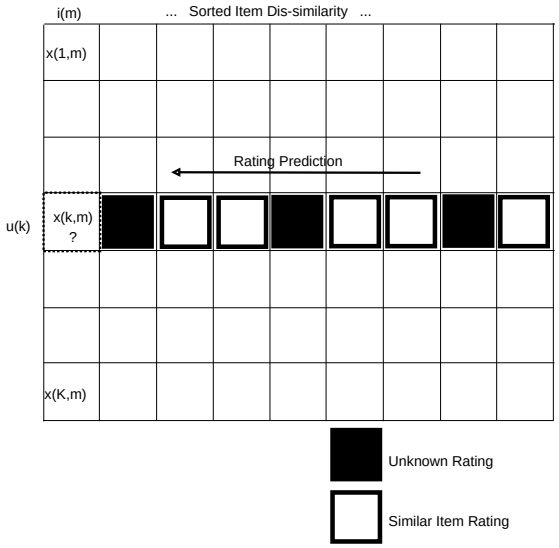

Figure 2.9: Rating prediction based on item similarity

2.3.4 Hybrid approach

As noted in section 2.3.2, several recommender systems use a hybrid approach by combining content-based and collaborative techniques. This helps to avoid certain limitations of content-based and collaborative filtering systems [5]. There are four main approaches for combining the two techniques into a hybrid recommender system.

Combining separate recommender systems

This approach implements content-based and collaborative techniques separately and combines their predictions [46, 15]. This can be done by combining the ratings obtained from individual recommender systems into one final recommendation, or by using the "best" individual system after measuring the quality of both systems.

Adding content-based characteristics to the collaborative approach

This approach incorporates some content-based characteristics into the collaborative approach. Content-based profiles, and not the commonly rated items, are used to calculate the similarity between two users. [46] makes it clear that this contributes toward overcoming some of the sparsity-related problems of a purely collaborative approach, since not so many pairs of users will have a significant number of commonly rated items. Another benefit is that users can be recommended an item not only when this item is rated highly by users with similar profile, but also directly, when this item scores highly against the user's profile.

Adding collaborative characteristics to the content-based approach

This approach incorporates some collaborative characteristics into the content-based approach. One example is to create a collaborative view of a collection of user profiles,

where user profiles are represented by term vectors [56]. This will result in an performance improvement compared to a pure content-based approach.

Developing a single unifying recommendation approach

This approach constructs a general unifying model that incorporates both content-based and collaborative characteristics. [6] proposes using content-based and collaborative characteristics, for example the age or gender of users or the genre of movies, in a single rule-based recommendation classifier.

2.4 Improving recommender systems

To provide better capabilities, recommender systems can be extended in several ways. Some improvements that have shown to give better recommendations will now be introduced.

2.4.1 Intrusiveness

Recommendations given by recommender systems are based on the user's opinion of the content. Opinions are expressed by ratings. Recommender systems are often distinguished by whether they operate on *intrusive(explicit)* or *nonintrusive(implicit)* ratings.

Intrusive rating refers to a user consciously expressing his or her preference for an item, normally in a *binary* or *numerical* scale. Using a binary scale, the user can only indicate whether he or she likes or dislikes an item, while using a numerical scale, the user can express the degree of preference for an item. One example of a system using intrusive ratings with a binary scale is Syskill & Webert [45], where users click on a thumbs up symbol when visiting a web site they like, and a thumbs down symbol when visiting a web site they don't like. The GroupLens system [48] is an example of a system using intrusive rating with a numerical scale. It uses a scale of one (bad) to five (good) for users to rate Netnews articles, and users rate each article after reading it.

Nonintrusive rating is done by interpreting user behaviour or selections to assign a rating or preference. Nonintrusive ratings can be based on browsing data in web applications, purchase history in web stores, or other types of information access patterns.

Even though nonintrusive ratings can be useful to limit the required user attention, they are often inaccurate and cannot fully replace explicit ratings provided by the user. Therefore, the problem of minimizing intrusiveness while maintaining a certain level of accuracy needs to be addressed by recommender system researchers.

2.4.2 Contextual information

Giving accurate recommendations is essential in recommender systems. Inaccurate recommendations will lead to displeased users, which will diminish the utility of the system. There are several recommender systems in both commercial and academic areas that deal with fixed user preferences. However, since the items preferred by a user may change depending on the context, these conventional systems have inherent problems. Contextual

information has therefore been used to improve the accuracy in recommender systems [44].

Dey defines context as any information that can be used to characterize the situation of an entity. An entity can be a person, place or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves [21]. In addition, Dey presents an architecture that supports the building of context-aware applications.

Contextual information can be crucial in some domains. The utility of a certain recommended item may depend on time and/or location. It may also depend on the person with whom the recommended item will be shared, and under what circumstances. A travel recommender system should not only recommend some vacation spot based on what this user and other similar users liked in the past. It should also consider the time of the year, with whom the user is travelling, and other relevant contextual information. Imagine a tourist having a cellular phone with a GPS[7] receiver. A recommender system could then be used to continuously send the tourist updated travel information that is relevant in terms of both time and location.

To be capable of using context in recommender systems, the content to be recommended needs some meta-data attached to it; something that describes the different contexts. This meta-data can be set manually or automatically by analyzing the content.

Manual modeling of content similarity

Meta-data can be added to content manually by the content providers, or by letting the users add this information while using the recommender system. One example of the first approach is how a movie can belong to a certain genre that is specified by the movie provider. The latter approach can be implemented by letting the users set certain movie properties. The movie recommender system developed by [24] even makes it possible for users to adapt new keywords describing a movie. These keywords are used to produce more accurate recommendations based on properties made by the users. The big advantage of this solution is that it adapts to changes regarding what the users find important, something which can change over time.

As stated in 2.3.1, manually categorization of content can be expensive, time-consuming, error-prone and highly subjective. Due to this, many systems aim at providing more automatic solutions.

Computational modeling of content similarity

Recently, much effort has been put in the area of automatic content similarity modeling within recommender systems. Some of this work has focused on the music domain. Research has usually tried to classify music by genre and artist, but also by contexts like mood.[22] focuses on music retrieval by detecting mood. In this project, mood detection is done by analyzing two music dimensions, tempo and articulation. Mood is divided

[7] Global positioning system

into four categories; happiness, anger, sadness, and fear.

In 2004, a world-wide cross-validation of music similarity systems was conducted, in the form of a public competition during the International Symposium on Music Information Retrieval (ISMIR)[8]. The result of this competition showed that it was possible to classify 729 music tracks into 6 different genres with an accuracy of 78.8%. They were also able to identify artists from a collection of 120 music titles out of a list of 40 artists with an accuracy of 24%.

MusicSurfer is a content-based recommender system that automatically extracts descriptions related to instrumentation, rhythm and harmony from music audio signals [14]. [36] developed a music recommender system for cars that can classify a wide range of stored music using automatic music content analysis. The system is able to extract some musical features from a CD without any prior information. This information is stored on a server together with the music. Users can then listen to music according to their current mood. The system also has the ability to give personal recommendations based on previously selected songs.

2.4.3 Evaluating recommender systems

Much effort has been put into the development of good metrics to measure the effectiveness of recommendations [29, 27]. In most literature, the evaluation of algorithms is done using *coverage* and *accuracy* metrics. Coverage measures the percentage of items for which a recommender system is capable of making predictions [29].

Accuracy measures can be either *statistical* or *decision-support* [29]. Statistical accuracy metrics compares the estimated ratings against the actual ratings. Techniques for doing this includes mean absolute error (MAE), root mean squared error and correlation between predictions and ratings. Decision-support measures determine how well a recommender system can make predictions of items that would be rated highly by the user. For example they include measures of *precision* and *recall*. Precision is the percentage of truly high ratings among those that were predicted to be high by the recommender system, while recall is the percentage of correctly predicted high ratings among all the ratings known to be high.

Although these measures are popular, they have certain limitations. One is that the users typically only rate the items that they choose to rate. The set of rated items will then probably give a fallacious view of preferences because users tend to rate the items that they like, not the items that they dislike. The consequence of this is that evaluation results only show how accurate the system is on items that users decided to rate, whereas the ability of the system to properly evaluate a random item is not tested. Another limitation with most of these evaluation metrics is that they do not capture the "quality" and "usefulness" of recommendations. Imagine a recommender system for a supermarket. Recommending obvious items such as milk and bread that the users are likely to buy, will give high accuracy rates. However, it will not be very useful for the

[8] http://ismir2004.ismir.net/

customer. It is therefore important to develop measures that also capture the business value of recommendations.

2.4.4 Other improvements

Other research issues within recommender systems include understanding of users and items [46, 47, 6], scalability [52, 53], explainability [30] and privacy [53]. Since these issues are outside the scope of this thesis, they will not be discussed.

2.5 Case study: Pandora vs. Last.fm

Music recommender services like Pandora[9] and Last.fm[10] have become very popular during the last years. These services are also called *personalized streaming radio stations*, because they allow users to specify a favorite artist, and then provide an Internet audio stream of similar music. Both provide the same service, but the underlying algorithms are different [38].

The recommendations produced by Pandora are based on inherent qualities of the music. When given an artist or a song, the service will find similar music in terms of melody, harmony, lyrics, orchestration, vocal character and so on. Pandora calls these musical attributes "genes". The database containing songs and genes belong to the "Music Genome Project". As explained in section 2.3.1, the main approach for producing recommendations based on content analysis is called content-based filtering. Pandora is an example of a service using this approach.

Last.fm is a social recommender, and knows little about the properties of each song. Instead it assumes that if the actual user and a group of other users enjoy many of the same artists, the actual user will probably also enjoy other artists that are popular in the group. Collaborative filtering systems produce recommendations based on the correlations between people with similar preferences. Last.fm uses this approach, but with a different focus than most of the other systems using this approach. Instead of focusing on improving the algorithms that are used to match similar users, Last.fm's innovation has been in improving the data that the algorithms work on. They claim that better algorithms are nice, but better data is nicer. An additional feature is an optional plug-in that automatically monitors different media-player software, so that user profiles gathered from these external resources can be included in the Last.fm application to give better recommendations.

2.5.1 Exploring new artists

Since both Pandora and Last.fm are services with the goal of helping their users discover new music, it is also important that the services keep themselves up to date. Adding new music to a service mainly involves two steps. First, the music and eventual attributes are added to the library. Then, the new music needs to be recommended before it can actually reach the users. This first step is said to be a bottleneck of content-based

[9] www.pandora.com
[10] www.last.fm

filtering systems, as explained in section 2.3.1. That is also the case for Pandora. For each song that is added to the library, employees at Pandora have to classify the song according to hundreds of musical attributes. This does not pose a problem for Last.fm, because here manual classification of each song is not required. However, Last.fm has its weakness in the second step. The sparsity problem, explained in section 2.3.3, is the problem of having a set of items to be evaluated that is larger compared to the number of evaluations given by the users. Last.fm may have difficulties letting new music reach the users because of this problem. Before new music can be recommendable, it needs time to get enough popularity to rise above the system noise level. Pandora does not have this problem because it is only comparing inherent qualities of the songs, not who they are popular with.

2.5.2 Overspecialization

Another issue that has to be taken into account in services like Pandora and Last.fm is the problem of overspecialization. In content-based services like Pandora, the system can end up *only* recommending songs that score highly against a user's profile. The user is then limited to being recommended items that are similar to those already rated. For example, a user with no knowledge about the music genre blues will never receive any recommendations for even songs that the user likes within this genre. One solution to this problem is to avoid recommending songs that are too similar to the songs already played. Pandora solves this problem by allowing the users to build several radio stations. Then it does not matter if one station is overspecialized as long as it is possible to create a new station.

For Last.fm, the problem is that recommender systems based on collaborative filtering tend to reward users who are similar to those who already use the system. If many of the users have the same taste as the actual user, the actual user will probably get good recommendations. If not, the actual user may get bad recommendations, and might end up not using the service. This is called a "locked loop" whereby the system only includes certain genres and styles. Although this may seem like a serious problem, a truly locked loop is unlikely for services like Last.fm, because of leakages. A group of users that share the same core musical tastes will have enough variance in secondary tastes to allow for a range of music that will always expand. However, the expansion will be slow for less popular genres.

2.5.3 Conclusion

Both Pandora and Last.fm are popular services and they both have millions of users. This shows that the services provide recommendations that satisfy the users. Which service to prefer is a difficult question, since they both have their strengths and weaknesses. Many will probably say that combining the approaches of the two services, would create the ultimate service, something which is not unlikely. It is said that Pandora is considered most promising in becoming the leading music recommender system [38], because it is easier for Pandora to incorporate Last.fm's collaborative filtering functionality, than the other way around. However, maintaining the manual work of classifying songs is expensive, and Pandora is probably not delivering proportionally more benefit for that cost.

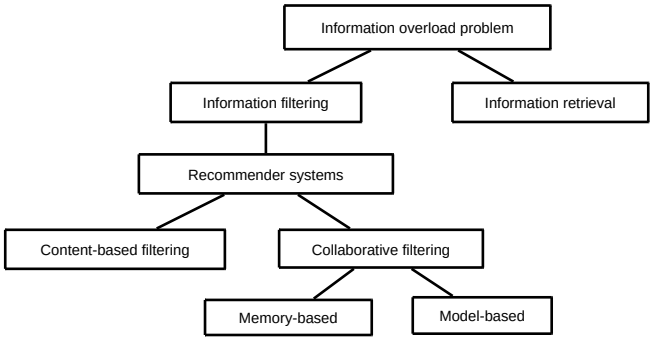

Figure 2.10: The information overload problem and a hierarchy of solutions.

2.6 Summary

This chapter first introduced the *information overload problem* in which users are finding it increasingly difficult to locate the right information at the right time. Then, a set of solutions to this problem has been presented. The hierarchy illustrated in figure 2.10 shows the relationship between these solutions. As can be seen, they either lie within the field of *information retrieval* or *information filtering*. While information retrieval systems filter information by letting users specify explicitly what information is needed, information filtering systems strive to adapt the users long-term interests and filter information based on user profiles.

Closely related to information filtering is the idea of having systems that act as personalized decision guides for users. These kind of systems are called *recommender systems*. There are mainly three algorithmic techniques for computing recommendations. *Content-based filtering* selects items to recommend based on the correlation between the content and the user's profile. *Collaborative filtering* chooses items based on the correlation between users with similar preferences. In addition, there exist *hybrid filtering* approaches that tries to avoid certain limitations related to content-based and collaborative filtering.

Furthermore, collaborative filtering are often classified as *memory-based* or *model-based*, which means that rating predictions are based on the entire collection of previously rated items, or on a model that reflects previously rated items respectively.

Various improvements to recommender systems have been introduced over the last years, and three of them are presented in this chapter. First, intrusiveness should be minimized while maintaining a certain level of accuracy, so that users can receive precise recommendations without being overly disturbed. Second, contextual information like time and location can be used to improve the accuracy of recommendations. Finally, various evaluation metrics have been developed to measure effectiveness and thus find out if a recommender system should be improved.

The last section of this chapter presents a case study where two popular music recommender systems are compared. Pandora and Last.fm may seem similar for the users, but the underlying algorithms vary. Pandora is mainly using a content-based approach while Last.fm mainly uses collaborative filtering techniques. In chapter 4, two filtering approaches that are similar to those used by Pandora and Last.fm, will be designed. The reason for presenting these systems is to give an example of two relevant commercial systems, and to explain some of the issues that have to be considered while designing such systems.

Chapter 3

Requirements

This chapter presents the requirements of our system, and is based on the problem definition in section 1.2. First an overview of the system model is presented.

3.1 System overview

In this thesis, a centralized recommender system for music is developed. An overview of the system is illustrated in figure 3.1. Clients communicate with a web server over the Internet. The web server provides a music service. After receiving song evaluations from the clients, the server will produce and provide the clients with personal music recommendations. Each recommendation consists of a play list with information about the music, and where the music is located.

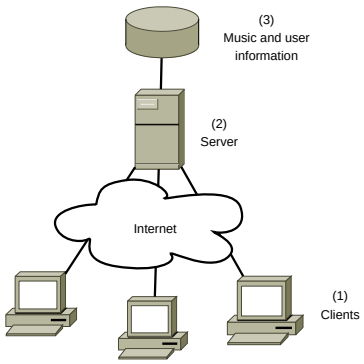

Figure 3.1: Recommender system overview.

3.2 Functional requirements

3.2.1 Client application

The client application is the link between the user and the server application. Its task is to gather information from the users and to allow users to play music. The information is sent to the server application, where it is stored, and later used to produce recommendations. In addition, the information is used to measure recommender precision. This allows for investigation of how precision is influenced by different recommender strategies. The requirements for the client application are:

R0 - Play music
The client application shall provide an interface that makes it possible to play music by selection, or by navigation through standard music player buttons like play, pause, stop and skip.

R1 - Request recommendations
The client application shall make it possible to request recommendations and to send the requests to the server application.

R2 - Evaluate songs
The client application shall make it possible to evaluate each song and to send this information to the server application.

3.2.2 Server application

The server application receives information from the client application, and provides the client application with recommendations. The requirements for the server application are:

R3 - Handle recommendation requests
The server application shall receive and handle requests for recommendations.

R4 - Store evaluations
The server application shall receive and store music evaluations.

R5 - Recommend using content-based filtering
The server application shall be capable of producing recommendations by interpreting the content and evaluations provided by the actual user.

R6 - Recommend using collaborative filtering
The server application shall be capable of producing recommendations by interpreting evaluations given by the actual user and other similar users.

R7 - Recommend using contextual collaborative filtering
The server application shall be capable of producing recommendations by interpreting contextual information given by the users, and evaluations given by the actual user and other similar users.

3.3 Non-functional requirements

R8 - Accuracy
The server application shall produce accurate recommendations that match the user's music preference.

R9 - Intrusiveness
The client application shall minimize intrusiveness and at the same time capture user attention so that an acceptable amount of evaluation data is received.

R10 - Scale potential
The recommender system shall have the potential of being scalable both with respect to size and geography.

Chapter 4

Design

This chapter presents our recommender system design and how to satisfy the requirements stated in chapter 3. First, the system architecture is explained, before discussing each system component in detail.

4.1 Architecture

4.1.1 Decomposition

The user interacts with the client application through the interface. The interface provides the user with the opportunity to play music (R0), request recommendations (R1) and evaluate songs (R2). Before music can be played, the client application needs a play list consisting of a set of recommended songs. This is provided by the server application after receiving a recommendation request (R3). To provide recommendations, the server application needs to produce recommendations based on evaluations received from the client application (R4). This can be done using content-based filtering (R5), collaborative filtering (R6) or contextual collaborative filtering (R7).

To provide a satisfactory service for the users, the recommendations must be accurate (R8), and the service should balance intrusiveness and getting a sufficient amount of evaluation data (R9). In addition, the system should have a scale potential, both when it comes to the number of client applications running simultaneously, and also when it comes to the geographical distance between a client and a server application host (R10).

A recommender system with the following components will be designed to conform to the requirements. All components are shown in figure 4.1:

Interface: Allows the users to interact with the system by playing and evaluating songs. It also allows users to request new recommendations.

Player: Plays songs from the music store.

Evaluator: Receives ratings and moods from the interface and sends this information to the server application. The evaluator in the server application receives

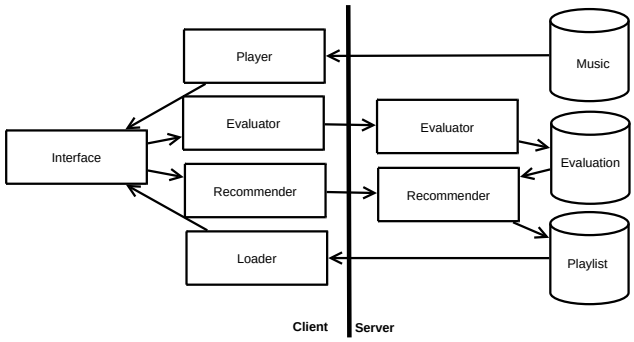

Figure 4.1: Client and server components.

ratings and moods from the client application, and stores them in the evaluation store.

Recommender: Receives requests from the interface, and sends the requests to the server application. Once the request is received by the server application, it produces recommendations and stores them in the play list store.

Loader: Loads information about the last set of recommended songs into the interface.

Music store: Provides the player with songs.

Evaluation store: Stores information about users, music and evaluations.

Play list store: Consists of a set of play lists containing the last recommendations for each user.

4.1.2 Scalability

Our design shows a typical centralized service in the sense that it is designed by means of only a single server application running on a specific machine. The main problem with this setting is that the server application simply can become a bottleneck as the number of users grow. Even with much processing and storage capacity, communication with the server application will eventually prevent further growth.

Scalability is a challenge that most commercial recommender systems have to deal with, since they may have thousands of users world-wide, requesting recommendations simultaneously. Because of this, scale potential will be discussed even though it will not be prioritized the development of our system.

4.2 System components

The components of our system will now be described in detail.

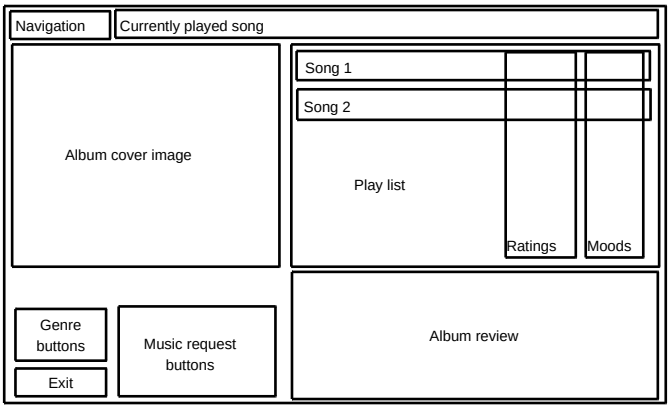

Figure 4.2: Interface sketch.

4.2.1 Interface

The interface allows users to interact with the system. We provide a GUI that makes it possible to play, evaluate and request music.

The interface receives music from the player, and provides the evaluator with evaluations given by the user. When the user wants more music, the interface is used to request new music, and the request is handled by the recommender. The interface receives information about recently recommended music from the loader.

Figure 4.2 shows a sketch of the interface that is designed. Playing songs will be done by either using the navigation buttons in the top left corner or by selecting songs in the play list. Each song will be evaluated by clicking rating buttons or mood buttons on each song in the play list. Recommendation requests will be sent by clicking the music request buttons, and genres will be changed using the genre buttons. In addition, album cover image and album review will be displayed. This will hopefully make the system more attractive for the test users.

4.2.2 Loader

The loader's task is to load a play list into the interface on start-up or after a new recommendation is produced. It uses the users id to locate the play list file for that user in the play list store. Then, it makes sure that the play list content is displayed.

4.2.3 Player

When the interface is loaded with a set of recommended songs, the player provides the interface with the actual music. It uses location information that the interface received from the loader to find music in the music store, and then downloads and plays each requested song.

While downloading and playing songs, the player may encounter delay problems. Delay

in wide-area networks may be due to scalability problems, with respect to size and/or geography. Scalability problems with respect to size may be caused by an overloaded server. Since our experiment will only include a relatively small number of users, we will probably not experience such a problem.

Geographical scalability problems may be caused by synchronous communication, where the client application blocks until a reply is sent back from the server application. In a wide-area system like ours, we need to take into account that communication may take hundreds of milliseconds. When our system in addition exchanges relatively large files (3-4 MB), some effort is required to make it scalable.

A general solution to scalability problems is to distribute and/or replicate server application and data. Replication not only increases availability to improve geographical scalability. It also helps to balance the load between the servers leading to better performance, and thus improves scalability with respect to size. A potential drawback with replication is that it may lead to consistency problems. Except for this drawback, replication is a technique that could help our system to scale.

In the case of geographical scalability, the general approach is to hide communication latency by introducing asynchronous communication. However, using asynchronous communication is usually not the best solution in interactive systems like ours, because the client application usually have nothing better to do than wait for an answer from the server application. In addition, our main concern is not the latency itself, but the problem of retrieving big files in a transparent manner. The user should not have to wait for the file to be downloaded before the corresponding song starts to play. Our solution is to stream songs. This makes it possible to play each song while the file is being downloaded. However, streaming may cause delay problems if download time is below a certain threshold. This threshold is given by the size of the file with respect to the time it takes to play the song. As long as the download time is low enough, so that each point of time in the song is downloaded before it is played, delay should not be a problem.

4.2.4 Evaluator

Each song in the play list can be evaluated using the interface. The evaluator enables the user to either rate or specify the mood for each song in the play list.

As explained in section 2.4.1, the user's opinion of recommended items is usually expressed explicitly by using a binary or numeric scale, or implicitly by monitoring information access patterns. We have chosen to use a combination of the two, based on a binary scale. Although a fine grained level of satisfaction will not be captured, we think that fewer alternatives will make it easier to use the system. The user can express satisfaction for a song by clicking a button pointing up or a button pointing down. In addition to this explicit rating, the system monitors the playing of each song. If a song is played through, the button pointing up is automatically set. If less than 30 seconds of a song is played, the button pointing down is automatically set. By using a combination of explicit and implicit evaluations, we hope to receive accurate user preferences while minimizing intrusiveness.

Mood specification is difficult to capture by monitoring information access patterns, and is therefore only done explicitly. For each song, the user can choose among the following moods: *angry, happy, relaxed* or *sad*. This set of moods may not be sufficient to give an optimal classification for all songs, but again we keep to our simplicity statement; fewer alternatives will make the system more user friendly. To ensure that all moods are covered, the selected moods represent the extreme points of what we normally classify as moods.

The evaluator in the client application receives evaluations from the interface, whether they are ratings or moods, and sends them to the evaluator in the server application. Each evaluation is sent at the moment the button is clicked, or automatically set. This allows for evaluation information that is always up to date, but may have the disadvantage of causing delay problems if the number of simultaneous users is high. A more scalable alternative would be to gather a set of evaluations and then send them together. When an evaluation is received by the server application, the evaluator in the server application stores the evaluations together with information about the actual user and item in the evaluation store.

4.2.5 Recommender

The recommender handles requests for recommendations. When the recommender in the client application receives a request for a new play list from the interface, the request is sent to the server application. A set of songs are then recommended based on data from the evaluation store and the selected filtering approach.

In general, a recommendation involves first producing a list of similar items or users, and then choose items to recommend based on this list. Our design employs a commonly used policy; only recommending items that the user has not evaluated. This reflects the key idea of many recommender systems; helping users find unfamiliar items that they probably will like. An alternative would be to include already rated songs, assuring that the user receives some songs that he/she is guarantied to like. Using this approach, the amount of evaluation data for our experiment would probably be reduced, because the recommender would recommend songs that have been already evaluated. This could weaken the result of our experiment, because of our already limited amount of evaluation data.

In section 2.5 we gave a case study with a comparison of two recommender systems that may seem similar on the surface, but where the underlying algorithm differ. Our design includes two filtering approaches similar to the ones used by Pandora and Last.fm respectively. In addition we have designed a filter that also considers contextual information, in this case user mood.

All filtering approaches produce recommendations by consulting the whole evaluation store database. An alternative and probably more scalable approach would be to compile a model for each user, based on the user's previous evaluations. Recommendations could then be produced more efficiently by consulting the model instead of the whole

database. However, this approach would have its drawback. Compiling the models can be time-consuming, and since a model has to be compiled before recommendations can be produced, this approach would prevent the production of up-to-date recommendations.

Content-based filtering

As described in section 2.3.1, content-based filtering is based on content analysis. Pandora is a system that uses this approach by manually classifying the songs by musical attributes. We have chosen to design a content-based filter that uses music genre as musical attribute, and provides the user with music within the same genres as the user previously has preferred. This may seem like a naive approach because music genres are usually not a specific way of categorizing music; one genre may contain a considerable variety of music. However, the main goal of this approach is to provide the users with a variety of music that more or less lies within the category that the users prefer, and then to gather as much evaluation data from the users as possible. The gathered evaluation data presents a picture of each user's preferences and therefore function as the user's profile. Profiles are needed as input to the other approaches.

First step in the content-based filtering approach is to compute a list of genres. These are the genres represented in the set of songs that the user has rated. For each genre, the difference between the number of positively and negatively rated songs is computed. This will make a genre distribution for the user, which is used to find out which genres the users prefer. Producing recommendations for a user is then done by selecting a set of songs that reflects the user's genre distribution.

Overspecialization must be considered in our design. When a user keeps rating songs within the same genre positively, the user may end up with a play list that only consists of songs within that specific genre. Getting the user out of this locked loop is done using two techniques; boosting songs from second choice genres, and allowing users to start over with a new combination of genres.

The first technique introduces some randomness in the recommendation process by boosting songs within genres that are not the user's first choice. Although this will allow the user to give positive ratings to songs that are not within the users favorite genre, it is not enough to get the user out of the locked loop. Imagine a user that has many positive ratings on songs within the rock genre. Even if the user gets some songs from the jazz genre recommended, and rates these songs positively, these few ratings will not be enough to get the user out of the "rock loop".

The second technique allows the user to select a combination of genres. The recommender component will then automatically feed a specific number of positive ratings into the system. These ratings reflect the genre combination that is selected. For example, if a user decides to listen to a combination of r&b and jazz, the user first selects these two genres. The user will then receive a play list where half of the songs are r&b and the other half are jazz songs. Songs recommended after an automatic feeding will only be based on fresh ratings; ratings given after the last genre selection. This gives a fresh start for recommendations, and lets the user out of the locked loop. The disadvantage

of this technique is that the user can loose an already built up profile by selecting a new genre combination. However, since the number of genres are limited and each genre is quite varied, it will not take much time to build a new profile using this approach. A better solution to the overspecialization problem might be to do like Pandora; offer several stations for each user to switch between.

The automatically generated ratings that are used when changing genre combination in this approach will not be considered in the other approaches, where ratings from this approach is used as input. Discarding these ratings will prevent the use of possibly false evaluation data, and therefore improve recommender precision.

Collaborative filtering

Collaborative filtering gives recommendations based on the correlation between people with similar taste. Two different approaches for doing collaborative filtering has been presented: memory-based and model-based. The former confronts the entire user preference database each time a recommendation is produced, while the latter confronts a smaller model that reflects user preferences.

Although the memory-based approach is relatively simple and have the advantage of avoiding an expensive model-building stage, it has some disadvantages. As pointed out in section 2.3.3 it requires computations that grow according to the number of users and items, and this is causing a scalability problem. In addition, the approach may have problems producing enough recommendations because the number of evaluated items is small compared to the total number of items. This is known as the sparsity problem, and may also cause problems with exploration new artists, as with Last.fm.

The model-based approach tries to solve some of the problems related to the memory-based approach by confronting a model for each user instead of the entire database. This will increase recommender performance if the number of users and items are high. However, the model-building stage is usually expensive, and freshness problems may be encountered if user preferences change fast with respect to the time needed to build the model.

In our design we have chosen to use the memory-based approach. Because of our limited number of users and songs, the scalability issue is not a problem. Sparsity may be an issue since the number of users and evaluations will be small compared to the number of songs. This may cause problems producing recommendations because of too few songs in the intersection of user profiles. We will try to solve this by first using the content-based approach to gather an acceptable amount of evaluation data before introducing this collaborative filtering approach. The model-based approach would be a wise choice if we had more users and scalability was a problem. Because of our small number of users, and the fact that user profiles are updated frequently, we prefer the memory-based approach to the model-based approach.

The process of producing recommendations can be divided into three steps:

1. Weight all users with respect to similarity with the actual user.

2. Select a subset of users to use as basis for recommendations.
3. Compute recommendations based on the subset of users.

The first step is to compute a ranked list of similarity weights to measure closeness between the active user and other users. Several different similarity weighting measures have been used.

Mean squared difference (MSD) is a similarity weighting algorithm used in the Ringo music recommender [55]. As the name of the algorithm implies, this similarity measure for two users is the average squared difference between ratings given by the two users on similar items. $msd_{a,u}$ is the mean squared difference between the actual user and user u.

$$msd_{a,u} = \frac{\sum_{i=1}^{m}(r_{a,i} - r_{u,i})^2}{m}$$

where m is the number of items that both users have rated, $r_{a,i}$ is the rating given by the active user on item i and $r_{u,i}$ is the rating given by the user u on item i.
We can see that $0 \leq msd_{a,u} \leq 1$. If both users have similar ratings for all items, $msd_{a,u}$ will be 0. If the ratings differ, $msd_{a,u}$ will be 1.

Grouplens [48] used the Pearsons correlation coefficient, which measures the degree to which a linear relationship exists between two variables. $pcc_{a,u}$ is the similarity weight between the active user and user u.

$$pcc_{a,u} = \frac{\sum_{i=1}^{m}[(r_{a,i} - \bar{r}_a)(r_{u,i} - \bar{r}_u)]}{\sqrt{\sum_{i=1}^{m}(r_{a,i} - \bar{r}_a)^2 \sum_{i=1}^{m}(r_{u,i} - \bar{r}_u)^2}}$$

where $\bar{r}_a$ is the average rating of the active user and $\bar{r}_u$ is the average rating of user u.

Other similarity measures includes the Spearman rank correlation coefficient and vector similarity cosine measure. Spearman rank correlation coefficient is similar to Pearson, but computes a measure of correlations between ranks instead of rating values. Vector similarity cosine measure has shown to be successful in information retrieval, but Breese found that vector similarity does not prove the optimal performance in collaborative filtering systems [13].

In our design we have chosen to use the MSD algorithm. It is intuitive, easy to test, and have shown acceptable performance [27].

In the second step we need to know the neighborhood of users that are helpful for making recommendations for the actual user. In theory, we could include every user in the database as a neighbor and weight the contribution of the neighbors accordingly, with distant neighbors contributing less than close neighbors. However, collaborative filtering systems are often handling many users. Making consideration of every user as a neighbor is therefore infeasible when trying to maintain real-time performance. A subset of all users must be selected as neighbors to guarantee acceptable response time. In addition, many of the users do not have similar tastes to the active user, so using them as neighbors will only decrease recommender accuracy. Two techniques, *correlation-thresholding* and

best-n-neighbors have been used to determine how many neighbors to select.

The first technique is used in Ringo [55] and sets an absolute correlation threshold, where all neighbors with absolute correlations greater than this threshold are selected. This ensures that neighbors have a minimum correlation value, but if the threshold is set too high, then very few users will be selected as neighbors.

The second technique, used by GroupLens [48], picks the best n correlates for a given n. This solves the problem with the first technique, but data from [27] shows that the value n affects the accuracy of recommendations. When using a small neighborhood, the accuracy suffers. The problem arise because even the top ten neighbors are often an imperfect basis for producing recommendations in recommender systems. This is because the user's different experiences result in many different subtleties of taste, and makes it very unlikely that there is a perfect match between users. Even if there is a perfect match, the evaluation data of less similar users will also be taken into account, and this might cloud the recommendations. Experience suggests that overall accuracy increases with increasing size of the neighborhood, and that this technique gives more accurate recommendations than correlation-thresholding.

Based on the results from [27], we have chosen to use the best-n-neighbors technique for selecting a neighborhood for each user. In our setting, we have a small number of users compared to what is usually the case in commercial recommender systems. The reason for doing neighborhood selection is therefore not to improve performance, but to filter out irrelevant users.

The third step in the collaborative filtering approach is to compute recommendations. Once a neighborhood has been selected, the ratings from users in the neighborhood are used to produce recommendations for the actual user. One way to do this is to traverse the neighborhood list starting with the most similar user. Songs that this user has rated positively and that the actual user have not yet rated, will then be recommended. Another approach is to make a ranked list consisting of items that have been rated positively by many similar users, and give recommendations based on this list.

We have chosen to use the first alternative because we conjecture that with our limited number of users no remarkable improvements will be gained by employing the second approach.

An optimization often used in recommender systems is rating normalization. Rating normalization copes with the problem of having users that do not rate on the same distribution, by performing some transformation to ensure that user ratings are in the same space. This technique improves recommender systems using a numerical scale, but does not have any effect on our binary scale. Hence, this optimization will not be considered.

Contextual collaborative filtering

As explained in section 2.4.2, context information has been used to improve accuracy in recommender systems. Meta-data describing context can be set manually by users, or

automatically by analyzing the content.

In this approach, mood is used as an additional filter variable in the collaborative filtering approach. This will probably increase recommender precision because the system will only recommend songs that are likely to reflect the user's mood at the time recommendations are requested. In the content-based filtering approach, songs were classified by mood, by letting the user provide real-time feedback. It is only in the contextual collaborative filtering approach that this classification is used in the recommendation process. When requesting recommendations, the user can choose between angry, happy, relaxed and sad. Given a certain mood, all songs classified with this mood will be used as input to the same collaborative filtering algorithm as described above.

Using the mood classification of all users as basis for recommendations can in theory result in recommendations that does not reflect all requested moods because users may have different opinions about what constitutes a mood. In addition, songs can be classified with the goal of preventing instead of supporting a certain mood. Some users may for example classify fast songs as sad because they want the music to make them happy. Others may classify slow songs as sad because they find this more comfortable when sad. In practice we think that the majority of the users will classify songs according to mood in a similar way, and that the inclusion of mood as an additional filter will result in more precise recommendations.

In addition, our conjecture is that context information is yet another attribute describing music. The difference between genre, that is used as an attribute in the content-based filtering approach, and mood that is used in this approach, may thus be subtle except that they represent different ways of classifying music. It is also likely that Pandora, described in section 2.5, uses mood as a musical attribute for producing recommendations, just as it uses genre. In our design, both genre and mood is set manually. The difference is that genre is set by the music providers and is used in a content-based filtering approach, while mood is set by the users and is used in a collaborative filtering approach.

After producing a set of recommendations, the information about the recommended songs will be stored in the play list store and the play list will then be loaded into the interface.

4.2.6 Music store

The songs that are recommended to the users are located in the music store. When a song is about to be played, the player retrieves the song by streaming it from the location that is specified in the play list.

In our design, both the music store and the rest of the server application reside on the same host. Since they are independent, an alternative could be to split and replicate the bottleneck of the two. However, they both represent resource-demanding components of the system, and replication of them both is probably needed to make the system more scalable. The server application includes the actual production of recommendations, something which requires much processing capacity while doing computations over

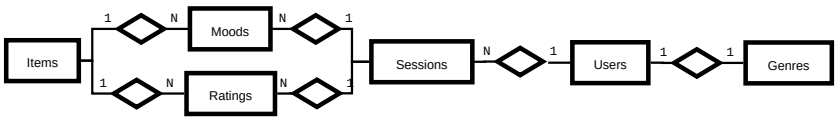

Figure 4.3: Evaluation store database.

the whole evaluation store. The music store is a network bottleneck because it needs to handle many simultaneously downloads.

4.2.7 Evaluation store

The evaluation store works as a storage for information about songs and users. In addition, it stores evaluations of songs provided by the users in form of moods and ratings. All this information is used by the recommender to produce recommendations. The evaluation store consists of the relation database illustrated in figure 4.3. It shows all entities and their relation with cardinality. The cardinality defines the numeric relationships between occurrences of the entities on either end of the relationship line.

The items entity does in our case represent songs and is connected to entities representing our two types of evaluation, mood and rating. One item can be related to many evaluations, while one evaluation can only be related to one item.

In addition, evaluations must be connected to users. Since each user will receive items in form of a play list, we have chosen to introduce an extra entity representing a session. A session is started when a new play list is created. As evaluations are a link between sessions and items, sessions are a link between users and evaluations. The sessions entity is connected to evaluations by letting one session relate to many evaluations, and one evaluation relate only to one session.

Sessions are connected to users by letting one session relate to one user, and one user relate to many sessions.

Finally, each user has a genre profile that reflects the user's current genre choice. The users entity is connected to the genres entity by letting one user relate to one genre and one genre relate to one user.

4.2.8 Play list store

The play list store is responsible of storing the last play list that is generated by the recommender for each user. The last play list for a specific user will be retrieved from the play list store when it is requested by the loader.

Each play list is stored as a file, containing information about each song in the play list. The information includes the name of song, album and artist, and the location of song, cover image and album review.

Our design is based on a server where each current play list is stored persistently, and reloaded upon startup. An alternative approach would be to use a server where the current play list is lost when the client application is closed, and a new play list must be created upon startup. Our approach has the advantage of giving the users more control over their play lists because they can choose to keep the current list as long as they want, even after restarting the client application. The server host will probably be heavily loaded while handling requests from users and producing recommendations. Introducing an extra task by letting the server application maintain a persistent play list for each user may therefore not be a good idea. A better solution would probably be to let the client host store the current play list persistently. One possible solution is to store the play list in the browser as a *web cookie*, just like web stores save the content of electronic shopping carts. This would probably reduce the load on the server host, and thus improve the performance.

4.3 Summary

This chapter presented the design of our recommender system. Its functionality includes different components that together shall fulfill the functional requirements stated in chapter 3. The *interface* allows users to interact with system, while the *player* makes it possible to play songs. The *evaluator* operates both in the client and the server application and allows for users to evaluate songs in their play lists by either rate or classify each song by mood. Evaluations are basis for recommendations requested by the client application and produced by the server application. Recommendations are produced by the *recommender*, which follows one of three filtering approaches. After producing recommendations, the corresponding play list is saved, and later retrieved by the *loader*. This component makes sure to retrieve the actual user's play list, and loads this into the interface. This is done on client application start-up or after producing new recommendations. Our design includes three persistent storages. The *music store* holds the actual music, the *evaluation store* holds user and evaluation data, while the *play list store* holds each user's play list.

Additionally, this design shall fulfill three non-functional requirements. Each filtering approach that is used by the recommender component shall result in *accurate* recommendations that suit each user's preference. Testing this design will include real-life users. Since user feedback is crucial for a good result, our design needs to make sure that test users are not unnecessarily disturbed while using the system. *Intrusiveness* therefore needs to be reduced. At the same time it is important to collect enough evaluation data. By using both implicit and explicit evaluations, we hope to balance the two. Although scalability is not prioritized in the development of this system, its scale potential has been discussed in this chapter. Our centralized architecture is probably not sufficient in a real-life setting, since commercial recommendation systems normally needs to handle thousands of simultaneous users world-wide. The scalability problem could probably be solved by replicating the server application and/or the music store. Scalability could also be improved by introducing a model-based filtering approach, so that recommendations are produced by consulting a model instead of the whole evaluation store database. By streaming songs, our design provides transparency in the sense that the user can play each song while the corresponding file is being downloaded.

Chapter 5

Implementation

This chapter explains the implementation environment in which the recommender system has been developed. Then, the implementation of each component designed in chapter 4 will be presented. Finally, the directives used to configure the system are listed.

5.1 Implementation environment

The implementation environment includes a server running the Unix-based operating system FreeBSD[1]. FreeBSD is known to be reliable and robust. It is also among the free operating systems that report the longest uptime[2]. Uptime is used to measure operating system reliability and shows how long the system has been "up" running. A long uptime indicates that the computer can be left unattended without crashing, and that no kernel updates have been necessary, as installing a new kernel requires a reboot and resets the uptime counter of the system.

An Apache HTTP server[3] is used together with the open source database MySQL[4]. They provide a secure and reliable platform for our system, and are also easy to set up locally in order to preview and test code as it is being developed.

The client application is implemented using Microsoft Windows XP[5] and Macromedia Flash 8[6]. To use the client application, the user will only need a web browser with Flash installed. Supported browsers are limited to Mozilla Firefox[7] and Opera[8].

[1] www.freebsd.org
[2] uptime.netcraft.com/up/today/top.avg.html
[3] www.apache.org
[4] www.mysql.com
[5] www.microsoft.com/windowsxp
[6] www.adobe.com/products/flash/flashpro/
[7] www.mozilla.com/en-US/firefox/
[8] www.opera.com

5.1.1 Programming language

For our recommender system we have chosen to use PHP[9] as programming language in the server application, and Flash/Actionscript in the client application. The reason for choosing PHP is its advantage of being a platform independent, server-side scripting language that is well compatible with Apache and MySQL. Because we want to provide the users with a music player application, a client application with an interface containing the functionality and appearance of an authentic music player needs to be implemented. Flash/Actionscript is chosen for the implementation of the client application because there already exists an open source project using this IDE to implement a basic web-based music player. Our client application therefore includes an extended version of this application.

5.1.2 Client application

The implementation of the client application is based on the XSPF web music player[10]. This player is part of a project within the sourceforge developer community[11], and provides a flash-based web application that plays MP3 files. The client application receives parameters in a URL in the following manner.

```
http://vserver.cs.uit.no/content_player.swf?host=http://vserver.cs.uit.no&bin=/moodbox&playlist=/playlists&user=412
```

vserver.cs.uit.no/content_player.swf is the location of the flash movie that forms the client application. The movie is a compilation of a flash file and an Actionscript file. The flash file includes graphic showing the appearance of the interface, and the Actionscript provides the functionality. There exist three variants of the client application, each corresponding to a certain filtering approach. The current file, *content_player.swf*, corresponds to the content-based filtering approach, while *cf_player.swf* and *mood_player.swf* correspond to the client application of the collaborative and contextual collaborative filtering approaches respectively. The reason for having three separate client applications is to maintain a structured and easily readable flash plan and Actionscript.

host is the name of the server that provides the server application. This URL is used by the Actionscript to connect to the server application.

bin is the server side directory for the binary files, and is used by the Actionscript to connect to the server application.

playlist is the server side directory where the play lists reside. By combining this directory with the user id, the Actionscript can find the location of the user's play list file.

user is the user id, and is used to identify the user requesting recommendations and giving evaluations.

When the client application is loaded, the parameters are set and the user can start to interact with the interface.

[9] www.php.net
[10] musicplayer.sourceforge.net
[11] sourceforge.net

5.1.3 Server application

The implementation of the server application differs from the client application in that all filtering approaches are integrated into one application. When the server application, that is located in *server.php*, receives a request from the client application, the server interprets the received variables and acts accordingly. The variables will now be explained.

> *option* indicates if the client application wants to set a rating or mood for a certain song, or request new recommendations.

The following variables are only used when the client application wants to set a rating or mood for a song.

> *sessionid* is the identification of a session for a certain user. A session is started each time a new recommended play list is produced.
>
> *itemid* is the identification of a certain song.
>
> *rating* is the chosen rating for a certain song. This is a number, 0 or 1, meaning negative or positive rating respectively.
>
> *mood* is the mood that a certain song reflects. This is a number 1-4, meaning *angry, happy, relaxed* and *sad* respectively.

The following variables are used when the client application requests new recommendations:

> *userid* is the identification of a certain user.
>
> *filtermode* is an indication of what filtering approach to use. The variable can be set to *content*(content-based filtering), *cf*(collaborative filtering) or *mood*(contextual collaborative filtering).
>
> *genrechange* indicate that the genre combination is changed, and is a variable used only in the content-based approach.

5.1.4 Communication

The Actionscript *LoadVars* class is used in the transferring of variables between the client and the server application. This class makes it possible to send all variables in one object to a specified URL, and to load all the variables at a specified URL into another object. The *LoadVars.onLoad* handler is used to ensure synchronous communication, which means that the client application continues only after the data is loaded.

All communication between the client and the server application is done using HTTP[12], where the client application initiates a request by establishing a TCP[13] connection to a particular port[14] on the server host. The server application is listening on that port and waits for the client application to send a request. Upon receiving a request, the server

[12]Hypertext Transfer Protocol
[13]Transmission Control Protocol
[14]Port 80 by default

Figure 5.1: User Interface using content-based filtering.

Figure 5.2: User Interface using collaborative filtering.

application accepts the connection and takes action according to the request before sending a response to the client application. HTTP is chosen as application layer protocol because it suites our needs and works well with today's Internet infrastructure.

5.2 System components

This section describes the implementation of the designed components. The components are shown in figure 4.1.

5.2.1 Interface

The interface component functions as a link between the user and other components in the client application by allowing the user to request, play and evaluate music. Our interface is a modified version of the interface in the XSPF web music player. This player

Figure 5.3: User Interface using contextual collaborative filtering.

initially offered streaming of MP3 files from one specific location. All functionality regarding the recommendation process is added to the original player. As shown in figure 5.1, 5.2 and 5.3, our interface differs depending on the filtering approach that is used.

Using the content-based filtering approach, the interface lets the user choose a combination of genres, and then receive recommendations based on the selected combination. In addition, the user can set both rating and mood for each song.

The collaborative filtering approach offers a more stripped interface where the users cannot choose genres or classify songs by mood.

The contextual collaborative filtering approach offers an interface similar to the collaborative filtering approach, but the user may in addition specify mood when requesting recommendations.

5.2.2 Loader

The loader parses the play list file and makes each song available through the interface. This is done by the following functions.

`playlistLoaded()`

This function parses the XSPF file that includes the play list for the actual user. For each song, the URL for the song, cover image and album review is extracted together with information about song, album and artist.

`addTrack(track_label, track_id)`

As the play list is parsed and information about each song extracted, the information is added to the play list that is displayed in the interface. This allows for the displaying and playing of each song. In addition, buttons that makes it possible to set rating and mood for each song is displayed.

5.2.3 Player

The player includes a set of functions that manage the playing of each song.

`loadTrack()`

This function is called each time a new song is about to be played, whether it is a result of using the play, next, or previous button, selecting the song from the play list, or letting the system play through the play list automatically. Before loading the new song, the playing time of the last played song is measured. If the last song was played for less than a number of seconds, and has not been rated while played, it will automatically get a negative rating. This is done because we conjecture that interrupting a song in the beginning may indicate that the user does not prefer the song.

Then, the next song will be loaded using the Actionscript function *loadSound(url, isStreaming)*. This function loads an MP3 file, found in the music store, into a *Sound* object. The *isStreaming* parameter is set to allow for streaming. Streaming makes it possible to play the song while the file is being downloaded. Each loaded file is saved in the browser's file cache. After starting to load a song, the cover image and album review are loaded and displayed.

```
stopTrack()
playTrack()
seekTrack(p_offset)
nextTrack()
prevTrack()
```

These functions are used to manage the navigation between songs implicitly by the system or explicitly by the user. Playing or stopping a song is done by starting or stopping the actual *Sound* object. After pausing a song, the song is resumed by seeking to the stored pause offset of the song. Skipping between songs (next or previous) is done by incrementing or decrementing the play list index variable. The actual loading of songs is always done by the *loadTrack()* function explained above.

5.2.4 Evaluator

An evaluation is a rating or mood given to a certain song by a user. The evaluator handles evaluations given by the users. It is split into a client evaluator and a server evaluator.

Client evaluator

The client evaluator's task is to capture the user's evaluation and send it to the server evaluator. This is done by the following functions. When the user is clicking a rating or mood button for a certain song, one of them is called.

```
sendRating(rate_mc)
sendMood(mood_mc)
```

Both functions translate the given evaluation from textual to numerical representation, and then send it to the server evaluator. After receiving feedback from the server evaluator, the number of evaluated songs in the current play list is calculated. This is done to assure that the user has evaluated a certain amount of songs before new recommendations can be requested. If the number of evaluated songs is over a threshold, a new play list can be loaded.

Server evaluator

When an evaluation is received from the client evaluator, the server evaluator stores or updates the evaluation in the evaluation store. The functions for handling evaluations are both part of the *Evaluator* class.

```
set_rating($sessionid, $itemid, $rating)
set_mood($sessionid, $itemid, $mood)
```

Both functions add or update the evaluation given by the user for a certain song. The *sessionid* reflects the play list where the song resides. Because each song is recommended once to each user, the set of play lists belonging to a certain user only contains unique songs. If any evaluation on a song is already set by the user, this evaluation is updated instead of added.

5.2.5 Recommender

The recommender is responsible of handling recommendation requests from the users and producing new recommendations. As with the evaluator, this component is split into a client and a server side component.

Client recommender

The client recommender receives recommendation requests from the users, and lets the loader insert the newly recommended songs into the play list that is displayed in the interface. The client side recommender consists of the following functions.

```
getList()
```

This function is called when the user requests a play list containing new recommendations. In the *content-based* filtering approach this can be done by choosing a genre combination and then click *Start*, or by clicking *Reload*. In the former case, the chosen genre combination will be sent to the server together with a *change genre* notification. In the latter, only a request for a new play list is sent. Using the *collaborative* filtering approach, new recommendations are received by clicking *Reload*. This will simply send a request for a new play list. In the *contextual collaborative* filtering approach, the request is done by clicking one of the mood buttons: *Angry*, *Happy*, *Relaxed* or *Sad*. Then, a notification indicating the selected mood is sent together with the request.

```
updateList()
```

After receiving feedback from the server side recommender, the client side recommender clears the current play list from the interface and asks the loader to retrieve the new play list.

Server recommender

When a recommender request is received from the client application, the server application will take actions based on the filtering approach used. The filtering approaches are content-based, collaborative and contextual collaborative. Each filtering approach will first produce a list of similar songs or users, and choose songs to recommend based on this list. Recommended songs are then registered in the user's play list. Because collaborative filtering requires a set of already existing user profiles to produce accurate recommendations, the two latter approaches use evaluation data from the former approach as input. The implementation of the server side recommender for the three approaches will now be explained further.

Content-based filtering

Before doing the actual recommendation (by producing a list of songs and then recommend new songs based on this list), the server application must check if the request received from the client application includes a *genre change* notification. As pointed out in section 4.2.5, users may change genres to avoid the overspecialization problem. If the *genre change* notification is included in the request, this indicates that the user has changed the combination of genres, and wants to receive recommendations based on this chosen combination.

The server application registers the new genre combination before invalidating previous ratings given by the user. This means that only later ratings will be taken into account when new recommendations are produced. Then, a specific number of positive ratings reflecting the chosen genre combination will automatically be inserted into the evaluation store, and related to the actual user. The first play list is therefore not a result of the user's own ratings. It only reflects the genres that are chosen by the user. Because these *automatic* ratings have not been set by the user, they should not be taken into account when the evaluation data from this approach is used as input to the other approaches. After invalidating old ratings and automatically inserting new ones, recommendations are produced. If the *genre change* notification is not set, the recommender starts producing recommendations directly without the invalidation and insertion stage.

The production of recommendations using this approach begins with computing the distribution of genres in the set of songs that the user has rated. This results in a list of genres and values indicating each genre's part of the songs rated by the user. Recommendations are based on this list. Each recommendation consists of a specific number of songs with the same distribution of genres as all *valid* songs previously rated by the user. Valid songs are the songs that are not invalidated after a genre change. Before producing the recommendations, ratings are retrieved from the evaluation store and checked against new recommendations to ensure that the same song is not recommended twice to the same user. Genres, represented in the user's genre distribution, that have a "part value" lower than a threshold, will have their value increased so that these genres become more dominating in the set of recommended songs. This is done to avoid that all recommended songs lies within the same genre, and thus reduces the overspecialization problem.

Collaborative filtering

When the server application receives a recommendation request, and the collaborative filtering approach is used, the recommender first produces a list containing a specific number of users that have more or less the same music taste as the actual user. Closeness between the active user and other users is calculated using the mean squared difference (MSD) algorithm described in section 4.2.5, and implemented using the following SQL query.

```
SELECT s2.userid as userid, COUNT(r2.itemid) as simcount,
SUM((r2.rating-r1.rating)*(r2.rating-r1.rating)) as spread
FROM ratings r1, ratings r2, sessions s1, sessions s2
WHERE s1.userid=$userid AND s1.sessionid=r1.sessionid
AND s2.sessionid=r2.sessionid AND r2.itemid=r1.itemid
AND s2.userid<>s1.userid AND s1.selected<=10 AND s2.selected<=10
GROUP BY s2.userid
```

This query takes as input the id of the actual user. The conditions

```
s1.selected<=10 AND s2.selected<=10
```

assures that *automatic* ratings are not taken into account. The query returns a table containing each user's id together with two values reflecting the dividend and the divisor of the MSD expression respectively. First, a value that we call *simcount*, telling the number of songs that both the actual user and other users have rated. Second, a value that we call *spread*, telling how many of these songs the two users have rated differently. From these values, the mean squared difference between ratings given by the active user a and user u on similar songs can be calculated.

$$msd_{a,u} = \frac{spread_{a,u}}{simcount_{a,u}}$$

After calculating the $msd_{a,u}$ value for a certain number of users, a similarity array containing users and their MSD value is created. The array is sorted by $msd_{a,u}$ starting with the lowest value, indicating the highest degree of similarity.

Recommendation production is done by consulting the similarity array. Starting with the first, and most similar user, all songs rated positively by this user will be retrieved. Before recommending a song to the actual user, it is checked whether the actual user has already rated this song, to assure that the same song is not recommended twice. When a specific number of recommendations have been produced, an array containing the id of each recommended song will be returned.

Contextual collaborative filtering

This approach uses the same algorithm as the collaborative filtering approach. The difference is that the server receives a notification about mood together with the recommendation request. Before finding similar users, this approach retrieves songs that are classified with the chosen mood. Then, users similar to the actual user will be identified in the same manner as in the collaborative filtering approach. Producing recommendations is done differently in the two approaches. In the collaborative filtering approach, all songs are considered when retrieving songs from similar users. Contextual collaborative filtering only considers songs with the mood specified by the actual user. Before adding a song to the array containing recommendations, mood is checked by ensuring that the song to be recommended resides in the set of songs with the specified mood. Only songs

matching the specified mood, will be recommended.

When a specific number of new songs have been recommended using one of the three filtering approaches explained above, a new session for the user is initiated and the play list for that session is generated and saved in the play list store. The loader will then retrieve the new play list and make it available through the interface.

5.2.6 Music store

The music store contains all the songs. Each song is represented as a MP3 file. In addition, the music store also contains a cover image and review of each album. These files are stored in JPG and TXT format respectively. The file hierarchy follows the artist/album/file structure, where a file can be song, cover image or album review. An example of this structure is shown below.

```
/music/Zero 7/When It Falls/
  cover.jpg
  review.txt
  Zero 7 - When It Falls - Home.mp3
  Zero 7 - When It Falls - In Time.mp3
  Zero 7 - When It Falls - Look Up.mp3
  Zero 7 - When It Falls - Morning Song.mp3
  Zero 7 - When It Falls - Over Our Heads.mp3
  Zero 7 - When It Falls - Passing By.mp3
  Zero 7 - When It Falls - Somersault.mp3
  Zero 7 - When It Falls - Speed Dial No. 2.mp3
  Zero 7 - When It Falls - The Space Between.mp3
  Zero 7 - When It Falls - Warm Sound.mp3
  Zero 7 - When It Falls - When It Falls.mp3
```

5.2.7 Evaluation store

The evaluation store's task is to store information about songs and users, and also the evaluations given by the users on specific songs. The relation database, illustrated in figure 4.3, shows the relation between different entities, and each relation's cardinality. Now, the evaluation store and the attributes of each entity will be explained in detail. Entities with attributes are shown in figure 5.6.

The items entity stores information about songs including song id and name of artist, album and song. In addition, it contains an attribute showing the music store URL where the song resides, and the song genre. Song genre is a number between 0 and 125 according to the ID3 format[15]. The genre is set by the music provider.

When the recommender is using the content-based filtering approach, only six different genres will be used: *Rock, R&B, Pop, Jazz, Folk* and *Other*. Each song in the music store is associated with one of these genres, according to its ID3 genre and this genre's similarity to the six genres that are used. The genre distribution and the number of songs within each specific genre is shown in figure 5.4. The genre distribution is also illustrated in figure 5.5. The distribution is rather uneven because *rock* and *other* have a relatively big share compared to the other genres. This is partly due to the classification of genres in general, where one genre may contain a considerable variety of music. *Rock* and *other* are examples of such genres. The use of this non-specific genre classification

[15] www.id3.org

Genre	*ID3Value*	*ID3Genre*	*Songs*
Rock	0	Blues	29
	1	Classic rock	1
	9	Metal	21
	17	Rock	1865
	86	Bluegrass	23
	5	Funk	30
	43	Punk	16
	131	Indie	431
Sum			2416
R&B	14	R&B	45
	16	Reggae	52
	42	Soul	78
	7	Hip-Hop	21
	15	Rap	12
Sum			208
Pop	13	Pop	462
	98	Easy listening	61
	116	Ballad	10
Sum			533
Jazz	8	Jazz	174
	71	Lo-fi	1
	10	New age	7
Sum			182
Folk	2	Country	127
	80	Folk	112
	88	Celtic	46
Sum			285
Other	12	Other	779
	20	Alternative	22
	52	Electronic	34
	255	Diverse	1563
	99	Acoustic	3
	24	Soundtrack	2
Sum			2403

Figure 5.4: The number of songs within each genre.

of songs may therefore weaken the content-based filtering approach by producing recommendations that do not necessarily lie within the genre that is specified.

Evaluations are split into moods and ratings which both function as a link between sessions and items. The attributes belonging to these entities are similar except from the fact that the moods entity stores values reflecting mood and the ratings entity store ratings. Since evaluations function as a link between sessions and items, the evaluation entities must include the primary key attribute of both the sessions entity and the items entity. Evaluations in form of moods or ratings are stored in an attribute containing an integer from 1 to 4 or 0 to 1 respectively.

The initiation of a session indicates the construction of a new play list. Sessions function as the link between users and their evaluations, and each session has an identification that is used as a link to each evaluation from this session. This link indicates which session each evaluation belongs to. Because each session is also linked to a certain user, the sessions entity also needs an attribute to identify users. Other information related to a session is the time when the session is created, the number of songs selected (included) in the session, an indication of the filtering approach used, and finally a boolean value telling if the ratings linked to the session are invalid. Invalid values are set to indicate which

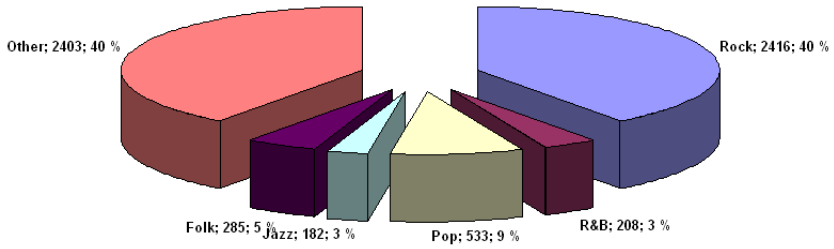

Figure 5.5: Genre distribution.

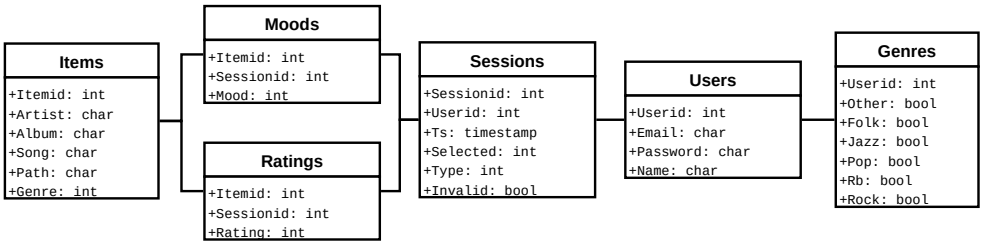

Figure 5.6: Evaluation store entities with attributes.

songs should be taken into account when recommendations are produced. This is done to reduce the overspecialization problem concerning the content-based filtering approach.

User data is stored in the evaluation store and are linked to both sessions and genres. Consistent linking requires that each user has an unique id. Email addresses could be used for this purpose, but since user id is often sent between the client and server application, an integer is considered more convenient. In addition information like email address, password and name are stored in its respective attributes. Email address and password is used for authentication.

Each user has a genre profile indicating the current genre combination. This information is only used by the content-based filtering approach and is stored in the genres entity. Each row in the entity consists of attributes identifying each user, and a set of boolean values reflecting the user's chosen genre combination.

5.2.8 Play list store

Each user has a file representing the user's play list. The file is stored in the play list store using a slightly modified version of the XML Sharable Playlist Format, XSPF [16].

```
<?xml version="1.0"?>
<playlist>
  <title>2051</title>
  <genre>rock</genre>
  <trackList>
    <track>
```

[16] www.xspf.org

```
      <id>35109</id>
      <location>http://vserver.cs.uit.no/music/Damien Rice/O/Damien Rice - O - Amie.mp3</location>
      <artist>Damien Rice</artist>
      <album>O</album>
      <song>Amie</song>
      <info>http://vserver.cs.uit.no/music/Damien Rice/O/info.txt</info>
      <image>http://vserver.cs.uit.no/music/Damien Rice/O/folder.jpg</image>
    </track>
  </trackList>
</playlist>
```

The play list for a user is stored in a file named by the user's id and the extension *.xspf*. Each time the user requests new recommendations, this file will be overwritten by the recommender and then reloaded and parsed by the loader. This file will always contain a set of tags identifying the current play list. The *Title* tag indicates the session id that this play list refers to. Session id is used by the evaluator to identify which play list the evaluated song belongs to. The *Genre* tag tells the last chosen genre combination. This text is used by the interface for display purposes. The main purpose of the play list file is to store the play list. This is a set of songs that the recommender has chosen based on the used filtering approach. The *trackList* tag contains the play list. Information about each song is stored under the *track* tag.

The *id* tag is the unique id of the song used in the evaluation store while the tag called *location* contains the music store URL of the song. The tags *artist*, *album* and *song* are used by the interface to display information about the song in the play list. *info* is the music store URL of the album review file, and *image* is the location of the file containing the album cover image.

5.3 Configuration directives

The directives that decide the system behavior will now be listed. They are all defined in the file called *config.php*.

FILTERMODE: Indicates which filtering approach to use. It can be *content* indicating content-based filtering, *cf* indicating collaborative filtering, or *mood* indicating contextual collaborative filtering.

HOSTURL: Tells the host where the server application resides. It can for example be *localhost* or as in our case *vserver.cs.uit.no*.

Database settings include:

DBHOST: Name of the host where the database resides.

DBNAME: Name of the database.

DBUSER: Database user name.

DBPASSWD: Password for the specific database user.

File settings specify naming of different files and folders including:

BINDIR: Folder name of the binary files.

PLAYLISTDIR: Folder containing the play list. This is the play list store location.

LOGFILE: File used for logging. Logging is done using the *dprint()* function that resides in *debug.php*.

COVERIMAGEFILE: Name of the file containing cover image for each album.

ALBUMINFOFILE: Name of the file containing review of each album.

Recommender settings specify values related to the production of recommendations. Some settings are used by specific filtering approaches, and some are used by all.

SIM_K: Number of similarities to return. This can be either similar items or similar users. Imagine that the collaborative filtering approach is used, and that this constant is set to 5. This means that when similar users are found, only the 5 most similar users will be considered when recommendations are produced.

REC_K: Number of recommendations to return. This constant specifies the maximum number of songs to be recommended each time recommendations are requested.

MOOD_K: Number of songs from a certain mood. Using contextual collaborative filtering, this constant tells the maximum number of songs with a certain mood that should be retrieved and sent as input to the collaborative filtering algorithm.

GENRE_K: Number of songs from a certain genre. Using content-based filtering, this constant tells the maximum number of songs with a certain genre combination that should be retrieved and sent as input to the content-based filtering algorithm.

RATING_K: Number of automatic ratings to insert when genre combination is changed using the content-based filtering approach.

5.4 Summary

This chapter has described the implementation environment, implementation of each system component, and configuration directives deciding the system behavior. The chosen programming language is PHP for the server application and Flash/Actionscript for the client application. The former is chosen because of its platform independence and compatibility with Apache and MySQL, while the latter is chosen because it easily can provide the functionality needed by a web-based music player. The client application receives parameters in a URL, while the server application takes action based on variables received from the client application. All communication is done using HTTP and the Actionscript *LoadVars* class.

Chapter 6

Experiment

This chapter contains a description of the experiments that was conducted for this thesis. A set of steps is followed in order to identify the right measure for our system, before the results of our experiment is presented. Throughout the chapter, the review of what has been done previously is separated from the analysis of our own system.

6.1 Measuring recommender systems

Recommender systems have been successfully measured by the following five steps [28, 29].

1. Identify the high-level goals of the system.
2. Identify specific tasks that are identified for the system.
3. Identify the dataset used for measuring the system.
4. Identify system-level metrics.
5. Perform experiment and measurement.

Each step will now be addressed and seen in relation to our system.

6.1.1 Identify goals

The goals of the system must be determined before any measurements can be done. Recommender systems are not valuable by themselves. Instead, they become valuable because they help users perform tasks better than the users would be able to without their assistance.

Before measuring a recommender system, it is important to identify which high-level goals the system should aim to achieve. Goals can for example be to improve economic, social or physiological values. It may also be to improve the quality of life for an individual. Although the different goals are generally measured separately, they may overlap.

The high-level goal of our recommender system is mainly to improve the quality of life for the users. The system does not provide any economic value or strengthen a social community. Instead it provides entertainment, recreation and automation by reducing unnecessary effort in search for information. We can measure if the system improves the quality of life by presenting the system to the users and observing the effects. High usage and stated user satisfaction indicate that the system is improving quality of life.

6.1.2 Identify tasks

Having specified the high-level goal of our recommender system, the next step is to identify the manner in which users will interact with the system. The choice of evaluation metrics will depend on the specific tasks that are identified for the system. These depend on the high-level goals of the users, and may include the following scenarios [29].

1. The user is about to make a selection that has significant cost and consequences, and wants to know the "best" option.

2. The user has a fixed amount of time and resources, and wants to see as many of the most valuable items as possible within that restriction.

3. The user wants to know about all relevant events within a specific content area.

4. The user wants to examine a stream of information in a given order, consuming items of value and skipping items of no value.

5. The user has a single item and wants to know if the item is worth consuming.

6. The user wants to locate a single item whose value exceeds a threshold.

Given that rating of songs is not crucial for the user's future, rules out scenario 1. This would be different if our system recommended e.g. insurance packages or other important items. Since the user normally does not have a fixed amount of time or resources while using our recommender system, scenario 2 is not relevant for this system. The users of our system are probably satisfied with receiving a play list containing a set of "decent" songs and do not need all relevant songs at the same time. This excludes scenario 3. Scenario 4 is generally meant for systems like bulletin boards, where some readers frequently examine the subject line of every article posted to a group. If a subject appears interesting, the entire article is retrieved and read. This functionality is outside the scope of our system approach, mainly because our system does not require all relevant items to be recommended in a strict ordering. Our system aims to identify a set of items that will be of interest to a certain user, and not to predict whether a particular user will like a particular item. Scenario 5 is therefore not relevant.

The user of our recommender system receives a play list upon request. This list consists of a specified number of songs that suites the user's taste. Because our system operates on similarity measure to decide which items to recommend, and because a small number of songs is sufficient, our system is closely related to scenario 6.

6.1.3 Identify dataset

Several key decisions regarding the dataset underlie the successful measurement of a recommender system. One important aspect is whether measurement is carried out *offline on an existing dataset*, or if it requires *live user tests*.

Much of the work in measuring recommender systems has focused on offline analysis of predictive accuracy [18]. Such measurements predict certain values taken from a dataset, and the results can be analyzed using the metrics discussed in the following section. Offline analysis has the advantage of being fast and economical, and can often utilize several different datasets and metrics at the same time [28].

However, offline analysis has two important weaknesses. First, the sparsity of ratings in datasets set limitations for the number of items that can be included in the analysis. It is infeasible to measure the appropriateness of a recommended item for a user, if the user has not rated this item in the dataset. Second, offline analysis cannot determine whether users will prefer a particular system over another. Preference can be due to its predictions, or to other less objective criteria as for example aesthetics of the user interface. Offline analysis is therefore limited to objective evaluation of prediction results.

Instead of measuring accuracy using offline analysis it is possible to conduct a live user experiment. Such an experiment allows for controlled and focused investigations of specific issues, for example testing well-defined hypothesis under controlled conditions. Field studies allow for a particular system to be made available to a community of users which is then observed to measure the effects of the system. This kind of measurement can capture additional data like satisfaction and participation.

Since we are interested in measuring our newly developed system, we need to gather data to use for our measurements. We are also interested in capturing user participation on the different filtering approaches. We have therefore chosen to conduct live user experiments using field studies to capture user participation and ratings. This also makes it possible to observe the effect of the system while it is used. An offline analysis of the data collected during the user experiment will then be performed.

6.1.4 Identify metrics

Throughout the years, recommender systems and other information filtering systems have been measured using a variety of metrics. The most commonly used metrics include *mean absolute error* and *precision and recall*. Each metric has its strengths and weaknesses with respect to the task that is identified for the system.

Mean absolute error

Mean absolute error (MAE) [28] belongs to a group of *statistical* accuracy metrics that compares the estimated ratings against the actual ratings. More specifically, MAE measures the average absolute deviation between a predicted rating and the user's true rating.

$$|\overline{E}| = \frac{\sum_{i=1}^{N} |p_i - r_i|}{N}$$

Since MAE treats the error from every item in the test set equally, this metric is most useful for scenario 4 where a user is requesting predictions for all items in the information stream. It is probably less useful for scenario 1,2,3 and 6. These scenarios represent tasks where a ranked result is returned to the user, and only the highest ranked items are relevant for the user. MAE provides the same weight to errors on any item. If certain items are more important than others, then MAE may not be the right choice.

However, evidence suggests that other metrics show improvements when the MAE value decreases. This shows that MAE should not be discounted as a potential metric for ranking-based tasks. In addition, its underlying computation is simple and has well studied properties.

Precision and recall

Precision and recall [28] belong to the group of *decision-support* accuracy metrics. Metrics within this group have in common that they determine how well a recommender system can make predictions of items that would be rated positively by the user. Precision and recall can be computed based on the table below [28, 29].

	$Selected$	$Notselected$	$Total$
Relevant	N_{rs}	N_{rn}	N_r
Irrelevant	N_{is}	N_{in}	N_i
Total	N_s	N_n	N

The item-set is separated into two classes, relevant or irrelevant. This requires the rating scale to be binary. Likewise, the item-set needs to be separated into the set that was returned to the user (selected), and the set that was not. We assume that the user will consider all retrieved items.

Precision is the ratio of relevant items selected to the number of items selected, and thus represents the probability that a selected item is relevant.

$$P = \frac{N_{rs}}{N_s}$$

Recall is defined as the ration of relevant items selected to the total number of relevant items available, and represents the probability that a relevant item will be selected.

$$R = \frac{N_{rs}}{N_r}$$

Recommendations produced by a recommender system are based on the likelihood that they will meet a user's taste. Only the users can determine if the recommendations meet their standards. Relevance in recommender systems is therefore highly subjective. Because of this, precision metrics may not be appropriate if users are rating items on a numerical scale with more than two ranks. Translation from a numerical to a binary scale is possible but difficult because the threshold determining if an item is relevant may vary for different users.

Recall may also be difficult to compute, because the number of relevant items in the

whole database must be determined. Since only the user can determine relevance, basically all users must examine every item in the database. If it is not important to consider all relevant items in the database, recall can be unnecessary. In this case, precision measure alone is probably sufficient.

One weakness of using both precision and recall metrics to compare different systems is that they must be considered together. The values representing precision and recall are also said do be inversely related and dependent on the length of the result list returned to the user [29]. If more items are returned, then recall values increase and precision values decrease. This requires that for one system, a vector of precision and recall values are needed to measure the system performance. This may complicate the comparison of many systems.

For scenario 1 and 2, precision and recall are not useful, because these scenarios represent tasks where ranked results are returned. Here, the user wants the current item to be more relevant than all other items with lower ranking. Precision and recall only measure binary relevance, and are not good at ranking the degree of relevance.

Using scenario 4 and 5, the usefulness of precision and recall depend on whether ratings are binary or not. If the system is producing non-binary predicted ratings, precision and recall may not be effective, because they can not measure how close predicted ratings are to real user ratings.

Precision and recall can be useful for tasks where there exists a clear threshold that divides relevant and irrelevant items. This is the case for scenario 3 and 6. Scenario 3 requires both precision and recall, but for scenario 6, precision alone is an appropriate metric, because the user requests only a subset of interesting items. Recall measure is not necessary because the system does not have to consider all relevant items in the database. Since our scenario is closely related to scenario 6, precision is considered a wise choice.

6.1.5 Perform experiment and measurement

A realization of our recommender system will have the goal of improving the quality of life, and will not provide any economical value or strengthen a social community.

The step of identifying the manner in which users will interact with the system, is related to our high-level goal. After listing different scenarios, we managed to find the one that is most similar to ours. Scenario 6 states that the user wants to locate a single item whose value exceeds a threshold, indicating that this item is relevant for the user. Our scenario is similar to this situation. The users are interested in receiving a list of songs that they probably will like. Unlike some of the other scenarios, the users of our system do not require all relevant songs in the entire database. They are satisfied as long as they receive a small set of “decent” songs.

The method of selecting the dataset for our measurement may affect our results. Considering offline analysis or liver user experiment, we chose to conduct live user experiments

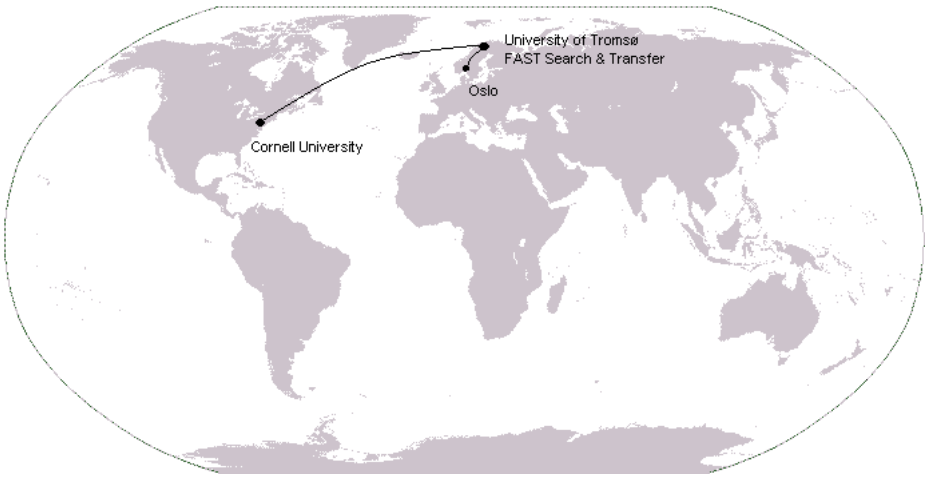

Figure 6.1: Location of the test users.

to capture data and observe the effect of the system while testing it. Measurement is then done by using offline analysis.

Which metrics to use for the measuring of a recommender system depend on the task identified for the system. For our scenario, precision is a good metric, because it measures the frequency with which our system makes correct or incorrect decisions about whether an item is relevant. Since our users do not need a complete list of all potentially relevant items, recall measure is not necessary.

Based on this discussion, we conjecture that recommender precision is a good measure for our experiments, and that this will be measured by first doing live user experiments to gather data and show user participation, and offline analysis to measure and present the actual precision data.

User experiments

User and rating data from our recommender system has been collected in live user experiments. This section will present these data to give a survey of the participation in the user experiment.

The system has been available to a community of users that was observed to ascertain the effect of the system. Most users are Norwegian men in their late twenties, studying or working at the University of Tromsø[1], at Cornell University[2], at FAST Search & Transfer[3], or in Oslo. The location of the test users is illustrated in figure 6.1.

The system was available for 33 days, from November 11 until December 19. During this period, all three filtering approaches were tested. The users had free access to the

[1] www.uit.no
[2] www.cornell.edu
[3] www.fastsearch.com

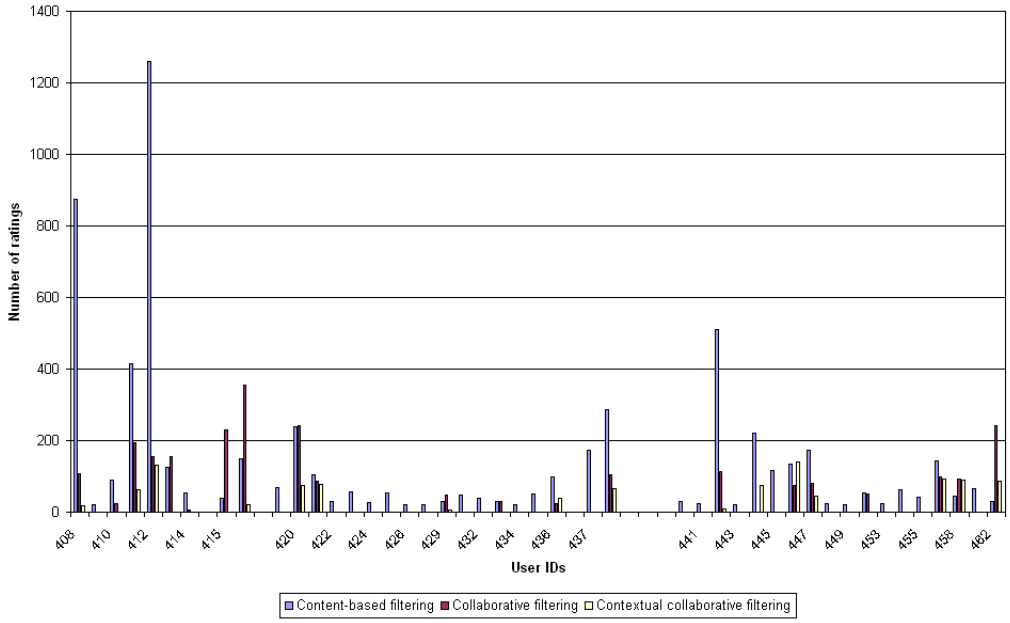

Figure 6.2: Ratings given by each user for different filtering approach in the whole period.

music player through a password-protected web-page[4]. By using the music player, users have requested, played and evaluated songs. The period and the number of days, users and ratings for each filtering approach is shown in the table below.

Filteringapproach	*Totalperiod*	*Days*	*Users*	*Ratings*
Content-based	20061115-20061203	19	45	6159
Collaborative	20061204-20061210	6	21	2515
Contextual collaborative	20061211-20061219	8	16	1034

The long content-based period of 19 days is due to the building of the user profiles. The profiles were needed in the collaborative and the contextual collaborative filtering approach. The number of days for the these approaches were 6 and 8 days respectively. Because of the profile-building stage, the content-based filtering approach had to be tested first. Collaborative filtering was tested before contextual collaborative filtering because the former were expected to give low precision compared to the latter. If the most precise solution was tested first, the users would perhaps stop using the system because of the decreasing precision. The table also reveals a decreasing number of both users and ratings throughout the whole period.

The number of ratings given by each user for each filtering approach is shown in figure 6.2. It shows that there are 5-10 users functioning as authorities. They have a much larger number of ratings than the other users, and they also tend to be the ones who have tested all filtering approaches. They therefore seem to be the loyal users of the system.

Figure 6.3 shows the number of ratings given by the users each day of the period. It

[4]Sharing of music has been approved by the University of Troms IT department

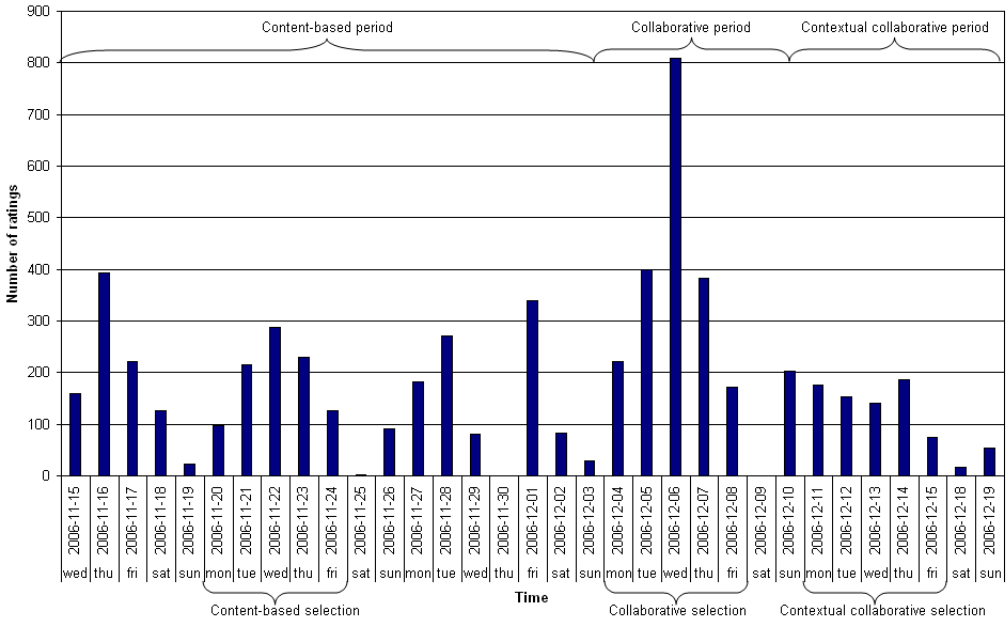

Figure 6.3: Ratings given by the users each day of the period.

shows that the number of ratings are decreasing with time. In addition, the number vary depending on the day of the week. The number of ratings are increasing until mid-week and then starts to decrease. This may seem surprising. One could think that people were listening more in the beginning and end of the week, since it is closer to the weekend, and people working and studying tend to be less concentrated these days.

Not surprisingly, the number is lower in the weekends than during the week. This is probably because the users tend to use the service while working or studying. If we base our evaluation on days with a low number of ratings, our result may be fallacious. As seen from figure 6.3, one period of weekdays is selected from the period of each filtering approach. These periods seem to be the best basis for our evaluation, because they had more ratings than other periods. In this way, we avoid erroneous conclusions due to basing our results on days with few ratings.

A comparison of the number of ratings for each filtering approach within the selected periods are shown in figure 6.4. It shows that during these periods, the number of ratings is clearly higher using the collaborative filtering approach, while the content-based approach usually is higher than the contextual collaborative approach. The selected periods, and the number of days, users and ratings for each filtering approach is shown in the table below.

Filteringapproach	*Selectedperiod*	*Days*	*Users*	*Ratings*
Content-based	20061120-20061124	5	26	2425
Collaborative	20061204-20061208	5	19	1814
Contextual collaborative	20061211-20061215	5	16	655

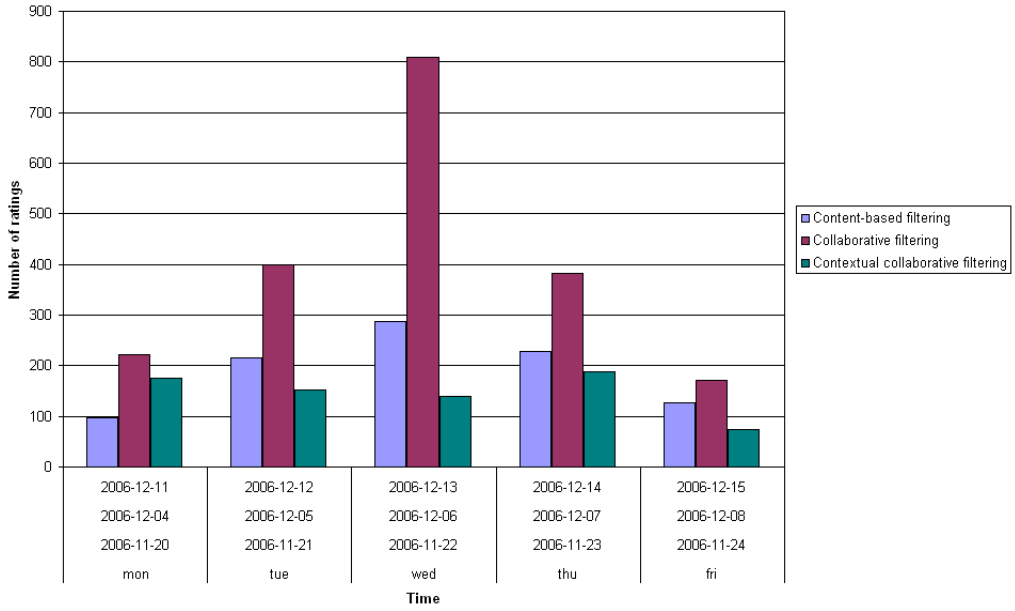

Figure 6.4: Ratings for each filtering approach in selected periods.

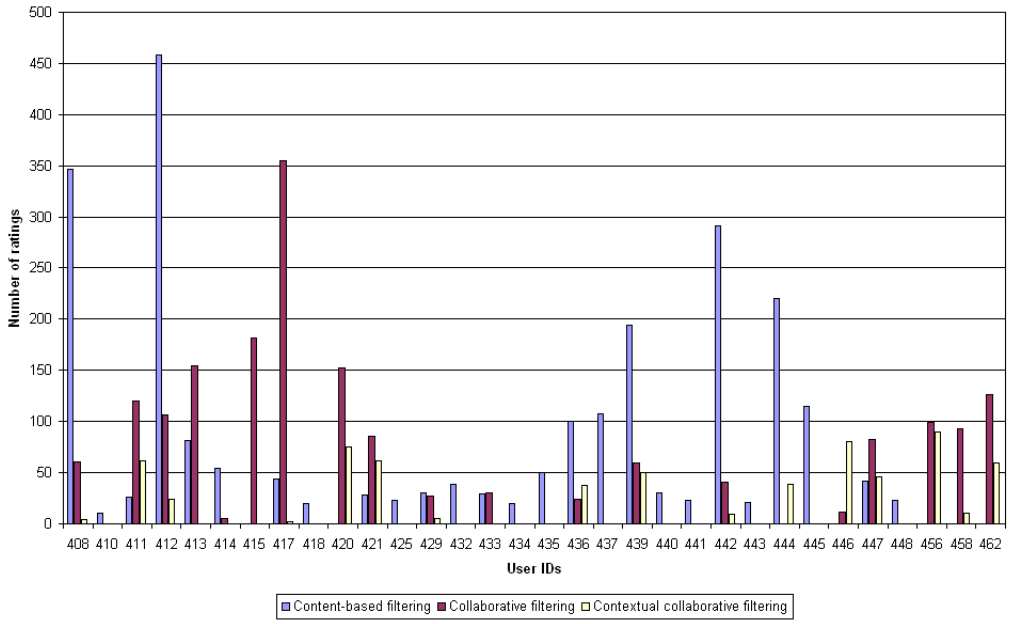

Figure 6.5: Ratings given by each user for different filtering approaches in selected periods.

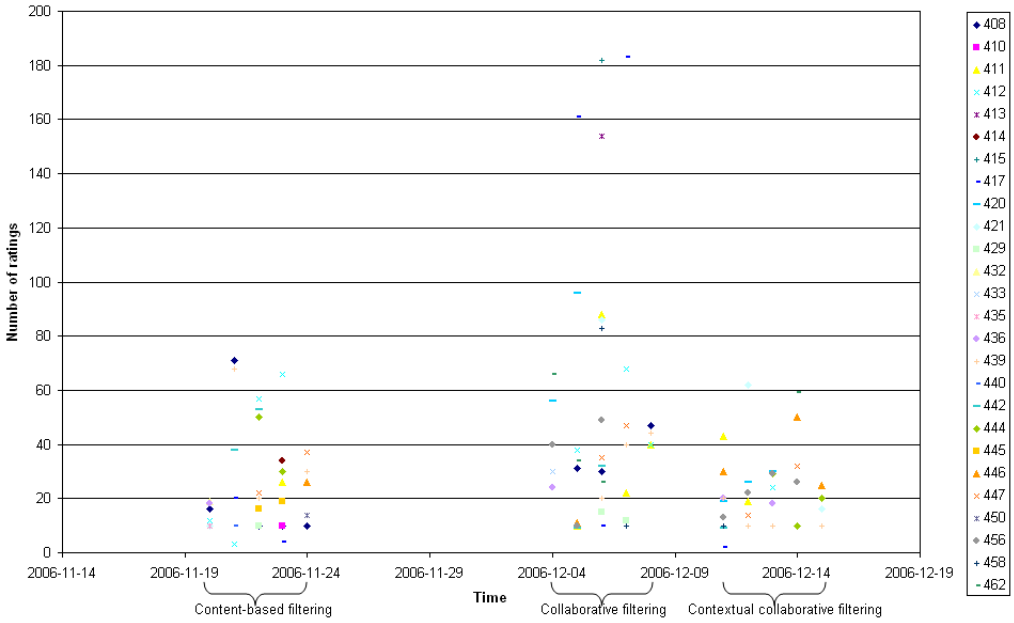

Figure 6.6: The number of ratings given by each user for each day in selected periods.

This table shows that the number of users and ratings have flattened out compared to the table showing the development of the whole period. Figure 6.5 also show this trend, by displaying the number of ratings given by each user in these selected periods. The reason for this is mainly because the content-based period has been cut down from 19 days to 5 days and therefore contribute with users and ratings accordingly.

Figure 6.6 shows the variation in the number of ratings given by each user for each day in the selected periods. Also here we can see the decreasing number of ratings, and also some of the authorities who have generally rated more songs than other users.

Studying these user/rating figures reveals some tendencies. First, the number of users and ratings are decreasing throughout the period. Second, it shows that the system has a few loyal users who have given many ratings and have tested all filtering approaches. However, most users have only rated some songs and have only tested one or two approaches.

Precision measuring

Based on the data that was gathered during the user experiment, the precision of recommendations produced by our recommender system has been measured using offline analysis.

Offline analysis has been done by performing calculations on gathered user and rating data. First, the precision of recommendations given throughout the whole period of user experiments was calculated using the following expression. It is a slightly modified version of the general precision expression that we conjecture will give a more intuitive presentation of recommender precision.

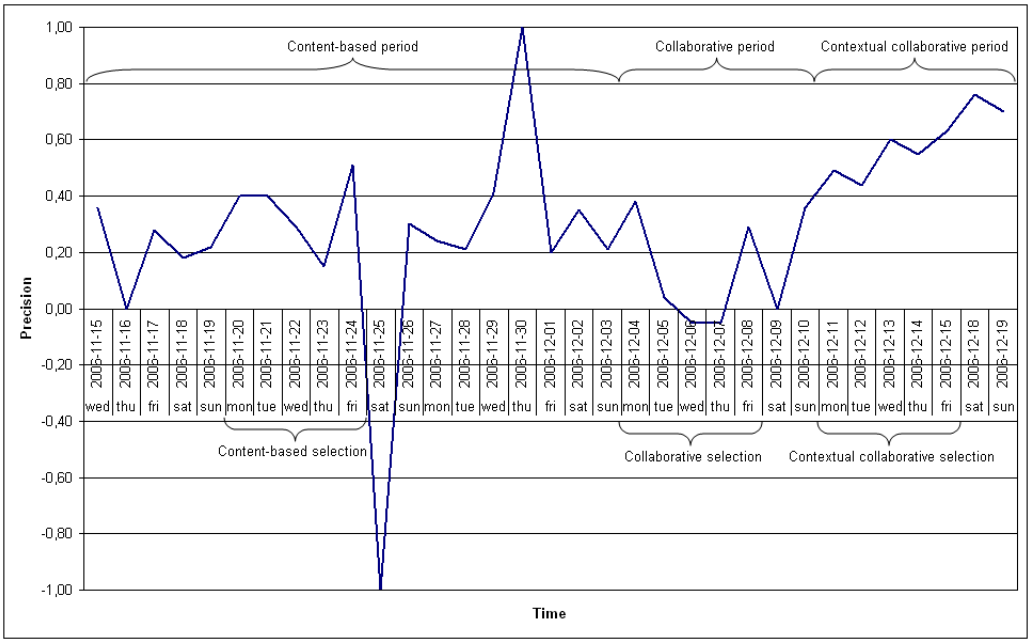

Figure 6.7: Daily recommender precision for each day in the whole period.

$$P = \frac{(N_{rs} - N_{is})}{N_s}$$

Instead of calculating only the number of relevant songs to selected songs, we have chosen to also include irrelevant songs in the expression. The general precision measure lets 0 indicate only negative ratings and 1 indicate only positive ratings. Our expression gives a value between -1 and 1, where -1 means that all songs have been rated negatively, 0 indicate a similar number of positive and negative ratings, while 1 means that all song have been rated positively. The drawback of not following the standard precision measure, is that our results are less comparable to other precision measures.

As an example, we have included an SQL query, calculating the daily precision values for the whole period.

```
SELECT DATE(s.ts) as day,
(COUNT(r.rating>0 OR NULL)-COUNT(r.rating<1 OR NULL))/(COUNT(r.rating)) as prec
INTO OUTFILE 'prec.txt'
FROM ratings r, sessions s
WHERE r.sessionid=s.sessionid AND DATE(s.ts)>='2006-11-15' AND DATE(s.ts)<='2006-12-19'
GROUP BY DATE(s.ts);
```

This query calculates the precision according to the expression explained above, and the result is shown in figure 6.7. The figure shows a big variation spanning from -1 to 1. However, most values lie between 0.20 and 0.60. The big peaks are mainly due to a small and therefore not representative set of ratings, often given during weekends. Otherwise, the figure shows an increase in precision at the end of the period. To investigate recommender precision further, we have extracted precision data from the same periods as selected earlier. The values for these periods are shown in figure 6.8.

The figure shows that the contextual collaborative filtering approach gives best precision

Figure 6.8: Daily recommender precision for each filtering approach in selected periods.

and that the content-based approach gives better precision than the collaborative filtering approach. All approaches seem to increase and decrease more or less simultaneously during the week, except for the contextual filtering approach. Its precision increases between Tuesday and Wednesday, while precision for the other two approaches decreases. The precision for all approaches are clearly increasing from Thursday to Friday. The figure also shows that collaborative filtering is the only approach to experience negative precision, meaning that the users dislike more songs than they like.

Having showed the precision for the different filtering approaches in selected periods, it may be interesting to see the precision for each user in the selected periods. This can be seen in figure 6.9 as it shows the variation in each user's precision for each day. This figure shows three groups of plots, each representing one selected period. Further, one can see an indication of where each group is centered. The first group, representing the content-based filtering approach, have some highly positive and some negative precision values, but the set of plots is centered around 0.4. The second group, representing the collaborative filtering approach, has a higher number of negative ratings, and this group is centered around 0.2. This shows that collaborative filtering generally gains lower precision than content-based filtering. Finally, the third group, representing the contextual collaborative filtering approach, shows a high number of highly positive precision values where some even equals 1. When precision equals to 1, this means that all songs have been rated positively. This group is centered above the other two groups, at around 0.6, showing that on average, more than 3/4 of the songs are rated positively.

In addition, this figure shows the pattern of how the different users are rating the songs.

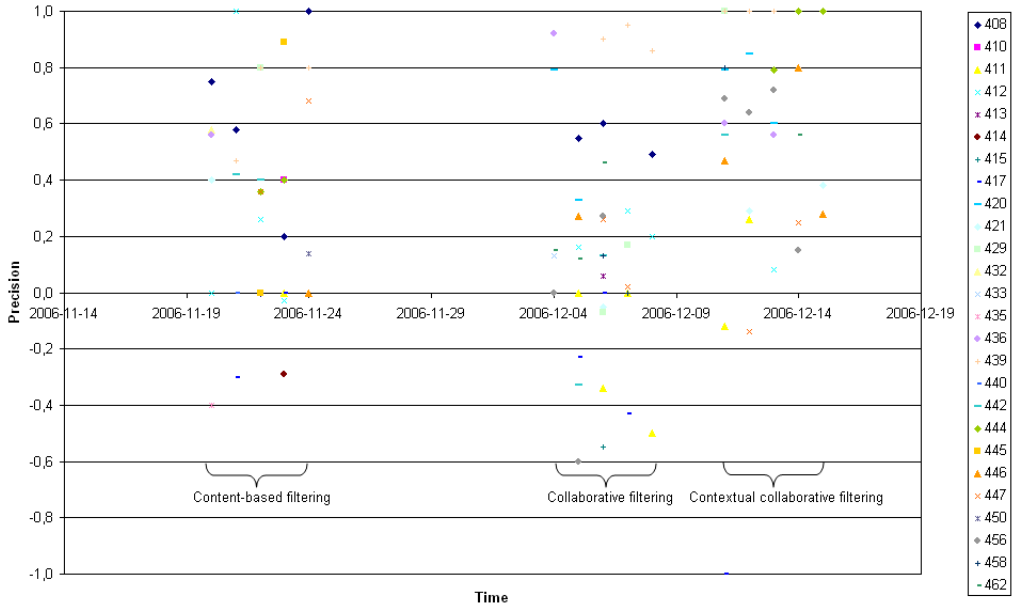

Figure 6.9: Daily recommender precision for each user in selected periods.

For example, some users tend to rate more positively than others, and some may gain increasing or decreasing precision during the whole period. User 439 and 408 seem to gain overall high precision, indicating that these users are usually satisfied with recommended songs. User 432 is generally less satisfied, and therefore gain lower precision. User 446 seems to gain increasing precision throughout the period, while 412 seems to gain decreasing precision.

6.2 Summary

This chapter has described the experimental part of this thesis. By following five steps for measuring recommender systems, we have identified certain properties of our system.

1. The high-level goal of our system is to improve the quality of life.
2. The task that is identified for the system allows the user to receive a list of songs that the user probably will like, without having to recommend all potentially relevant songs in the entire database.
3. The dataset that is used for the measurements is captured using a live user experiment. Then, an offline analysis is performed on the dataset.
4. The metric that is best suited for the measuring of our system is precision.

The result of the user experiment shows that the number of users and ratings are decreasing throughout the period, and that the system has been used by 5-10 authorities who have tested all three filtering approaches, and contributed with many ratings. The

majority of the users have tested one or two of the filtering approaches and have given few ratings.

The result of the precision measurement shows that the precision differs for the three filtering approaches. Contextual collaborative filtering gives best precision, while content-based filtering gives better precision than collaborative filtering. The precision for all three approaches seems to follow more or less the same pattern during the week. Finally, the pattern of how users rate differ. Some users are steadily gaining the same precision level, while others increase or decrease during the test period.

Chapter 7

Evaluation

We shall now evaluate the recommender system that has been developed, and find out if it meets the requirements specified in chapter 3.

7.1 Functional evaluation

R0-R7 state requirements that concern the functionality of our recommender system. In rough terms, the system shall allow users to play music (R0), request recommendations (R1, R3, R5, R6 and R7) and evaluate songs (R2 and R4).

Missing functionality

Playing songs and navigating in the play list works fine, although some functionality have been requested. First, some users have asked for a way to fast-forward/rewind songs. Others have complained about the placement of the navigation buttons, saying that a more natural placement would be on the bottom of the player. Some users have said that the rating buttons are misleading. Instead of using triangles pointing up or down we should use symbols showing a thumb pointing up or down. Another thing that may weaken the usability of the player is a small user interface with small buttons. Although the balance between size and usability was considered throughly while developing the system, some users may disagree in the chosen solution.

Play list advantage

A functional advantage of our system compared to other related systems is the possibility to select songs to play from a list of recommended songs. Other commercial music players like Pandora and Last.fm only recommend one song at a time, and does not allow to start playing the same song over again. For example, if the stop button is clicked in Last.fm while listening to a song, the song disappears. Then, if the play button is clicked, another song is recommended. Our system does on the other hand always keep the current play list, allowing the user to freely select what songs to play, even after restarting the client application. This gives more control to the user, but it includes the disadvantages of having a stateful server. First, this may be a burden for the server host because of additional computations that follows the maintaining of state for each user.

Second, the fact that our centralized server is a single-point of failure would get even bigger consequences as the server is stateful. If the server host crashes, the play-list of all users would be lost. However, since all play lists are stored persistently on the hard drive and not in memory, a crash would normally not ruin the data. Although the problem of having a stateful server is not serious in our setting, it would be if our system was part of a commercial service handling thousands of users. One solution to the problem could be to store each play list at the client host, for example in the browser as a *web cookie*.

Caching problem

While doing the user experiment, some users experienced problems refreshing their play list upon recommendation requests. Flash movies (swf files), like other documents and media retrieved by a web browser, are often saved, or cached, locally on the user's hard drive. The next time that media is requested, the web browser may load the file from the cache instead of downloading it over the network. This might be desirable for a flash movie whose content does not change often, but undesirable for movies like ours, that are updated frequently with new content.

The problem can be solved easily by manually clearing the browser cache. However, an automatic solution to this problem is required to maintain the availability of the system. Using the following techniques, flash movie files can be forced to expire immediately from the web browser cache. First, the *Expires* header of an HTML document telling the web browser when a cached document should expire from the cache can be used. Using a date in the past ensures that the document always will be expired. Second, using the *Pragma: No-Cache* header directs the browser to not cache the document at all. Finally, it is possible to force the linked page to be loaded from the server and not from the browser cache by simply placing a pseudorandom number at the end of the query-string. Since some technique's functionality may depend on the browser that is used, all techniques were used to avoid caching of play lists.

Halting problem

A problem that has been experienced by one user testing our recommender system is that the client application stopped playing songs when its window resided behind other application windows. In this case, the user had many applications running simultaneously, and when the player window was given focus after stopping, the song continued at exactly the same point as it stopped. Without investigating this phenomenon further, it may seem like the process running the client application got such low priority when it got in the background that it stopped playing.

After all, our recommender system satisfies all the functional requirements stated in chapter 3 (R0-R7).

7.2 Non-functional evaluation

R8-10 states the non-functional requirements of our recommender system. Each one will now be evaluated.

7.2.1 Accuracy

R8 states that the server application shall produce accurate recommendations that match the user's music preference.

Precision

Our system has been measured by calculating the precision of recommendations produced by the system. Recall has not been considered because the users do not need all relevant songs when requesting recommendations. As long as they receive some songs that meets their needs, they will be satisfied. However, if our system for example allowed the users to explicitly request songs from specific artists or albums, then recall would also be important to assure that all relevant songs were considered.

User influence

Our experiment shows that collaborative filtering gives a low precision compared to content-based and contextual collaborative filtering. The collaborative filtering approach differs from the other approaches by not letting the users make any choices about what songs to receive by providing realtime feedback. Recommendations are only based on the user's, and other similar users previous ratings.

Unlike collaborative filtering, content-based filtering allows the users to make choices, more specifically by selecting a preferred genre combination while requesting songs. Contextual collaborative filtering allows the users to make choices by receiving songs based on their specified mood. Giving the users more influence, by for example indicating context in the process of recommending songs, may seem to provide more precise recommendations. This is clearly demonstrated from our experimental results. Collaborative and contextual collaborative filtering use the same algorithm, except that the latter includes mood as an extra filtering attribute. The experimental result shows that contextual collaborative filtering gains far better precision than collaborative filtering. This states the fact that contextual information improves recommender precision.

Attribute quality

The experimental result shows that contextual collaborative filtering outperforms content-based filtering when it comes to precision. Recall that they both do filtering based on attributes describing the content; contextual collaborative filtering filters on mood and content-based filtering filters on genre. The result may therefore indicate that collaborative filtering as filtering technique outperforms content-based filtering. However, this fact may be concealed by the attributes that are used during the filtering process. Without having done any further investigation, we conjecture that songs are specified more accurately with respect to mood, than genre. The unfair genre distribution shown in figure 5.5, and the fact that songs are explicitly, and therefore relatively accurately related to a mood, supports this assertion. Since the way songs are specified with respect to musical attributes may differ, our experimental result may give a false impression about the precision obtained by content-based and contextual collaborative filtering. A better way to compare the two approaches would be to let them both filter on the same musical attribute. Then, we would probably get a more reliable result.

Decreasing precision

It has to be mentioned that the precision measure for content-based and contextual collaborative filtering are based on a smaller dataset than the precision measure for collaborative filtering, making the precision of the former approaches less reliable, and maybe falsely positive. In addition, further investigation of recommendations produced by the collaborative filtering approach shows that they initially descend from users that have some ratings in common with the actual user. However, after a few recommendation requests, the number of similar users starts to decrease and results in mostly random recommendations. This is probably due to our limited number of users, making precise recommendations disappear faster than new ones are produced.

Figure 6.7 may prove this tendency showing a very steep decrease in precision in the period of the collaborative filtering approach. It is clear that our system suffers from a sparsity problem by having to few users. This is a known problem for collaborative filtering systems. We tried to reduce the problem by having a long profile-building stage using the content-based filtering approach. However, the sparsity problem makes it difficult for our system to produce precise recommendations in the long run.

Disadvantage of large profiles

Although we conjecture that building large and specific user profiles would improve recommender precision, our experiment showed something else. Instead of selecting similar users with large profiles in the recommendation process, our system preferred small similar user profiles. This happened because the actual user had more ratings similar to small profiles than to big profiles. Large profiles seemed to specific to be used as basis for recommendations. It would be interesting to investigate this further and see if an experiment with more users would make the system utilize more of the large profiles.

Unique preferences

Another group of profiles suffering in our system is the group of profiles reflecting unique music tastes. While using collaborative or contextual collaborative filtering, our system tends to reward users who are like those who already use the system. If there already exist many users with similar taste in respect to the actual user, the actual user will probably get precise recommendations. If not, the user may receive less precise recommendations. The effect of this is that the user will probably stop using the system. Evaluation of our system shows that some users have tested the system more than others. These users have normally also tested more than one filtering approach. This may indicate that they have been more satisfied with recommendations than other users, and that they constitute the "main-stream" group of users in the system. Other users may have stopped using the system because they received recommendations that did not suit their more unique taste. Providing all users with precise recommendations is therefore a challenge that must be faced, especially within commercial recommender systems, as they are influenced by a big competition of capturing users having a big variety of taste.

New users problem

Another weakness of our system is that it is not possible for new users to join the system while using the collaborative or the contextual collaborative filtering approach, simply because there does not exist any profiles for these users. This problem must be solved in commercial systems, where new users are joining the system continuously. More complex hybrid content-based/collaborative filtering solutions solves this problem, by letting recommendations be based on a rich set of content-describing attributes in addition to ratings given by the user and other similar users.

Filter overkill

As building more complex systems normally is done to increase recommender accuracy, this may not be the best solution to improve the overall quality of the system. As mentioned in chapter 6, precision and recall is inversely related, meaning that prioritizing precision will decrease the probability of having relevant songs recommended. We tested our contextual collaborative filtering approach using genre as an additional filtering attribute. The result of this intensive filtering was that the system returned a couple of songs before it ran out of recommendations. This demonstrates that accuracy should not always be the main goal while developing recommender systems.

After all, the server application in our recommender system produces enough precise recommendations to demonstrate that the tested concepts work satisfactory. This means that R8 is fulfilled. However, to provide precise recommendations over time, our system would have needed more users.

7.2.2 Intrusiveness

R9 states that the client application shall minimize intrusiveness and at the same time capture user attention so that an acceptable amount of evaluation data is received.

Capturing enough user feedback usually requires that the user must be disturbed. Since recommender systems requires feedback from the users to provide personalized recommendations, and that disturbing the users too much would probably make them stop using the systems, balancing disturbance and data capturing must be considered in most recommender systems.

Mixing explicit and implicit ratings

A normal way of balancing the two includes a mixture of explicit and implicit user ratings. Explicit ratings requires disturbance because the user must for example click a button to express preference. Because the users have full control over given ratings, this will give precise feedback. Implicit ratings do not require the user's attention because preferences are expressed by monitoring the user's behavior. This will give less precise feedback since the user is not aware of ratings that are given. Our system uses a mixture of explicit and implicit ratings by letting users both click an up or down button on each song, and at the same time sets the up button if the song is played through and the down button if less than 30 seconds of the song is played. This works well although especially implicit feedback requires some tuning to work optimally.

Problem remembering songs

Some users have expressed that after continuously playing a set of songs, they have forgotten whether they liked the songs played early in the set. They therefore had to go through the songs over again to find out if they liked them, and then set ratings accordingly. The main problem with this was that the system did not provide a way to fast-forward/rewind a song. This should be allowed since deciding whether a song is preferable usually requires that the user starts listening some seconds out in the song.

Another way to make it easier to rate a set of recently played songs could be to start playing a small summary of all songs after capturing the user's attention. The user could then easily determine which song to rate positively and which songs to rate negatively.

Capturing user attention

Since user attention is important to receive user feedback, and thus produce accurate recommendations, the user interface of our system is used for providing the user with an image of the album cover and an album review related to most songs in the music store. This makes the interface look more attractive and may thus result in a better user experience.

Another technique used by our system is to stop the player after playing five consecutive songs, and display a message asking if the user wants to continue playing. This is done to assure that the user is testing the system and to avoid playing songs if the user has put away the ear-phones. Continue playing songs without the user listing would cause false implicit ratings and would decrease recommender precision. This technique seems to work well although some users have complained about being interrupted while playing music.

Binary vs numeric rating scale

Our system is using a binary rating scale, which does not give the same degree of rating accuracy as using a numeric rating scale. Binary ratings was chosen because our conjecture was that few alternatives would make rating easier for the users, something that probably would increase the number of ratings given by the user. It is uncertain if our solution works better than a numerical scale. Some users have complained about the binary scale and its disadvantage of not giving the possibility to express accurate preference for songs.

One possible solution that gives more degrees of preference, and at the same time keeps on to the binary scale, is to let explicit and implicit ratings give different rating weight. This can be done by for example having a scale from 1 to 4, where 1 is the most negative rating and 4 is the most positive. If the user rates a song explicitly by clicking the down button, this song gets a rating equals 1, and if the song is explicitly rated by clicking the up button, this will yield a rating equals 4. Implicit ratings will be given depending on whether less than 30 seconds of a song is played or the whole song is played through. The former case will give a rating equals 2 and the latter will give a rating equals 3. Having a

finer grained rating scale where explicit ratings are weighted more than implicit ratings will make user preferences more detailed, and probably recommendations more accurate.

Coarse-grained mood alternatives

Our system's mood classification may suffer from the same problem, namely giving too coarse-grained alternatives for the users. The current alternatives for classifying mood only represent the most extreme degrees of mood without giving the possibility to express moods "in between". Extreme cases were chosen because we want to reduce the number of alternatives, and instead of missing some alternatives it is better to give alternatives that cover all possibilities.

After all, our mixture of explicit and implicit ratings makes sure that the system minimizes intrusiveness and at the same time captures user attention so that an acceptable amount of evaluation data is received. R9 is therefore fulfilled.

7.2.3 Scale potential

R10 states that the recommender system shall have the potential of being scalable both with respect to geography and size.

Geographical scalability

Our system is tested from different locations, as shown in figure 6.1, and with a different number of users playing music simultaneously. When a certain song is requested, the download time is shown using a download bar in the interface, in the background of the box showing the currently played song. The faster this bar expands, the faster the song is downloaded. After using the system from different locations, it is experienced that the download time depends on location in respect to the server. However, the download time has never been so low that delay has been experienced. Although, most users have been located within the same region as the server, some users have tested the system from more distant places like the east coast of the United States. The fact that no delay has been experienced by these users indicates that the system is geographically scalable with the current number of users.

Size scalability

Considering the number of users, and scalability with respect to size, our result is more uncertain. With our small number of test users, the system seems to perform satisfactory. On the other hand, if the number of users were to increase, communication delay would probably be considerable, causing problems communicating with the server application and the music store. A solution to this problem could be to conduct standard scaling techniques like distribution and replication. In a large setting with many users world-wide, it would probably be necessary to replicate the server application to improve availability and reliability all over the world, or at least in the part of the world where most of the users reside. Since music taste differs depending on location, the music store should be distributed so that for example most Asian music reside on servers in Asia while most Latino music reside on servers in the south of Europe and South America. To provide availability and reliability, each server should also be replicated depending on

the server load.

After all, our system has the potential of being scalable if some scalability techniques are deployed in both our server application and music store. R10 is therefore fulfilled.

7.3 Summary

In this chapter we have evaluated our recommender system against the requirements stated in chapter 3. We conclude that all requirements, both functional and non-functional, have been fulfilled.

The system functionality is acceptable although some features like fast-forward/rewind each song and playing a summary of songs in a play list have been requested. In addition, the users have experienced some problems refreshing their play list. One advantage of our system is that it lets the user receive a list of songs, and not only one at a time. Our system also lets the user keep each play list as long as the user wants, even after restarting the client application. This user control is not provided by most other commercial music services.

The most important of the non-functional requirements concerns accuracy. Without accurate recommendations we would not need a recommender system in the first place. Giving random suggestions, like many non-personalized streaming radio stations do, could be sufficient. Our recommender system does like most other similar systems require input from users to produce accurate recommendations. Our experimental result shows that users become more satisfied with recommendations if they are able to make choices about what songs to receive, in our case by choosing music genre or mood. Although musical attributes like genre and mood may help producing more accurate recommendations, the performance of the filtering approach that is used also depend on how accurate each song is specified with respect to the musical attribute. Our experiment also shows that recommender precision decreases and that this happens because precise recommendations disappears faster than new ones are produced. This is due to our limited number of users. Having big user profiles does not seem to be an advantage in our setting, where these profiles become too specific. Finding similar profiles therefore becomes difficult. Our system also suffers from a problem of not accepting new users when the collaborative of contextual collaborative filtering is used. This is due to a property of pure "social" recommender systems requiring an existing profile to receive recommendations. Finally, we have found out that practicing extensive filtering does not necessarily improve the overall quality of the recommender system, as too much filtering will reduce the number of recommended songs.

Our system strives to minimize intrusiveness and at the same time capture enough user feedback by utilizing a mixture of explicit and implicit ratings. This seems to work fine although especially implicit feedback requires some tuning to function optimally in our system. To avoid receiving positive implicit ratings while the user for example puts away the ear-phones, our system stops playing after five consecutive songs. Some user have mentioned that they have forgotten whether they liked a song after playing a set of songs. A possible solution is to play a summary of all songs in the play list for example when

the user requests a new play list. This will help users to express their preference for each song in the play list. Our system uses album information to help capturing user attention. Our way of capturing rating data can probably be improved for example by introducing a more fine-grained rating scale. This would make user preferences more detailed and probably make recommendations more accurate. The coarse-grained alternatives for selecting moods is due to the desire of having few alternatives and at the same time cover all subtleties of mood.

Considering our system's scale potential is important because today's commercial recommender systems usually provide services to thousands of users world-wide. Since we throughout this thesis have compared our system to such services, scalability should also be discussed, although it has not been of main importance in our development. We conclude that scalability with respect to both geography and size is not a problem in our current system, but with more users over a large distance, our system would suffer. A solution could be to selectively replicate our server application and selectively distribute and replicate our music data.

Chapter 8

Conclusion

This chapter summarizes our thesis by first concluding whether our achievements have fulfilled the problem definition. Then, future work is presented.

8.1 Achievements

The problem definition from section 1.2 is stated below.

> *This thesis shall focus on development and evaluation of a recommender system within the music domain. Different approaches for computing recommendations will be designed, implemented and tested with real end-users. Evaluation will be done by assessing the system functionality and comparing the recommender precision obtained by each approach.*

We have developed a recommender system that allows users to play music, request recommendations and evaluate songs. Three different filtering approaches have been implemented and tested. First, a content-based filtering approach producing recommendations by interpreting music genre and evaluations provided by the actual user. Second, a collaborative filtering approach producing recommendations by interpreting evaluations given by the actual user and other similar users. Finally, a contextual collaborative filtering approach producing recommendations by interpreting mood information given by the users, and evaluations given by the actual user and other similar users.

Experiments have been done by conducting live user experiments to capture user and evaluation data. This data has been used as basis for offline analysis measuring recommender precision for the different filtering approaches. The result of the precision measuring shows that the precision differs for the three filtering approaches. Contextual collaborative filtering gives best precision while content-based filtering outperforms collaborative filtering.

Our conclusion from the experiment can be split into four parts.

1. Realtime feedback increases recommender precision. By allowing the users to choose music genre or mood while requesting music, they are more likely to receive accurate recommendations.

2. User mood has proved to be an important aspect of the users context. Our experiment has shown that by integrating a mechanism for mood filtering into the recommender system, it has been possible to produce recommendations that better suits the users often varying music preferences.

3. Our recommender system has proved to be useful within the music domain. By producing recommendations based on user feedback, our system has been able to satisfy the users. By using the best filtering approach, on average, 3/4 of the recommended songs were positively rated.

4. Precision measuring alone has been sufficient to analyze our system. Since the users are satisfied with a small set of songs that they probably will like, and do not require all relevant songs, recall measuring is not necessary.

8.2 Future work

In this section we present issues that should be explored to improve our recommender system.

Improve filtering approaches: Our filtering approaches have been developed with the goal of demonstrating the concepts behind each approach. Further improvements can be done on each approach. One example is to introduce more coarse-grained classification of music, for example by artist and album, instead of only songs.

Hybrid solution: Combining content-based and collaborative filtering would reduce sparsity-related problems including the problem of including new users in the system.

Contextual information: Filter on other types of contextual information, for example information describing time, location or what the user is doing.

Safe user feedback: Allowing feedback from the users to change the behavior of the system so that the system only recommends "safe" songs that the users are guaranteed to like. Imagine a user who is in party mood, and makes the system aware of this. The system then recommends all the user's favorite party songs.

Community: Each user profile could be part of a community with similar profiles where users for example could recommend music directly to each other. In addition, other services could be included in the system, like discussion forums and chat rooms.

Other media: The system could be expanded by offering other types of media like pictures and movies.

Scalability: Modify our system to function better in a bigger setting, with many users world-wide.

Testing: Additional testing over a longer period and with a larger variety of users would verify our results and the tendencies shown in the experimental results. Also, testing of a baseline system that only recommends random songs would prove the effect of using a recommender system.

Bibliography

[1] *Internet usage statistics - The big picture of world Internet users and population stats.* http://www.internetworldstats.com/stats.htm. [cited at p. 1]

[2] *Architecture of the World Wide Web, Volume One.* W3C Recommendation, http://www.w3.org/TR/webarch/, 15 December 2004. [cited at p. 5]

[3] *Interview with Jeff Bezos.* BusinessWeek, http://www.businessweek.com/ebiz/9903/316bezos.htm, March 16th 1999. [cited at p. 8]

[4] *Fourth Workshop on the Evaluation of Adaptive Systems in conjunction with UM'05.* http://www.easy-hub.org/hub/workshops/um2005/challenge.html, 2005. [cited at p. 2]

[5] G. Adomavicius and A. Tuzhilin. *Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions.* IEEE Transactions on Knowledge and Data Engineering, http://dx.doi.org/10.1109/TKDE.2005.99, 2005. [cited at p. 16]

[6] C. Basu, H. Hirsh, and W. W. Cohen. *Recommendation as Classification: Using Social and Content-Based Information in Recommendation.* AAAI/IAAI, citeseer.ist.psu.edu/basu98recommendation.html, 1998. [cited at p. 17, 20]

[7] N. Belkin and B. W. Croft. *Information filtering and information retrieval: two sides of the same coin?* ACM Press, http://doi.acm.org/10.1145/138859.138861, 1992. [cited at p. 7]

[8] T. Berners-Lee. *Information Management: A Proposal.* CERN. World Wide Web Consortium (W3C), 1989. [cited at p. 5]

[9] T. Berners-Lee, R. T. Fielding, and H. Frystyk. *RFC 1945: Hypertext Transfer Protocol - HTTP/1.0.* 1996. [cited at p. 6]

[10] T. Berners-Lee, R. T. Fielding, and L. Masinter. *RFC 2396: Uniform resource identifiers (URI): Generic syntax.* August 1998. [cited at p. 6]

[11] T. Berners-Lee, L. Masinter, and M. McChahill. *RFC 1738: Univorm resource locators (URL).* 1994. [cited at p. 6]

[12] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau. *Extensible Markup Language (XML) 1.0 (Fourth Edition) - Origin and Goals.* World Wide Web Consortium, http://www.w3.org/TR/2006/REC-xml-20060816, 2006. [cited at p. 6]

[13] J. S. Breese, D. Heckerman, and C. Kadie. *Empirical Analysis of Predictive Algorithms for Collaborative Filtering.* citeseer.ist.psu.edu/breese98empirical.html, 1998. [cited at p. 13, 14, 15, 36]

[14] P. Cano, M. Koppenberger, and N. Wack. *Content-based music audio recommendation.* MULTIMEDIA '05: Proceedings of the 13th annual ACM international conference on Multimedia, http://doi.acm.org/10.1145/1101149.1101181, 2005. [cited at p. 19]

[15] M. Claypool, A. Gokhale, T. Miranda, P. Murnikov, D. Netes, and M. Sartin. *Combining Content-Based and Collaborative Filters in an Online Newspaper.* citeseer.ist.psu.edu/article/claypool99combining.html, 1999. [cited at p. 16]

[16] C. Cleverdon. *The Cranfield tests on index language devices.* Aslib Proceedings 19, 1967. 173-192. [cited at p. 9]

[17] D. W. Connolly and L. Masinter. *RFC 2854: The 'text/html' Media Type.* 2000. [cited at p. 6]

[18] B. J. Dahlen, J. A. Konstan, J. L. Herlocker, N. Good, A. Borchers, and J. Riedl. *Jump-starting movielens: User benefits of starting a collaborative filtering system with.* [cited at p. 12, 57]

[19] Peter J. Denning. *Computing as a discipline.* http://doi.acm.org/10.1145/63238.63239, 1989. [cited at p. 2]

[20] M. Deshpande and G. Karypis. *Item-based top-N recommendation algorithms.* ACM Trans. Inf. Syst., http://doi.acm.org/10.1145/963770.963776, 2004. [cited at p. 14]

[21] A. Dey. *Understanding and using context.* Personal and Ubiquitous Computing, Vol 5, No. 1, pp 4-7, citeseer.ist.psu.edu/dey01understanding.html, 2001. [cited at p. 18]

[22] Y. Feng, Y. Zhuang, and Y. Pan. *Music Information Retrieval by Detecting Mood via Computational Media Aesthetics.* http://ieeexplore.ieee.org/iel5/8792/27823/01241199.pdf, 2003. [cited at p. 18]

[23] N. Fuhr and C. Buckley. *A Probabilistic Learning Approach for Document Indexing*, volume 3. Information Systems, citeseer.ist.psu.edu/fuhr91probabilistic.html, 1991. 223-248. [cited at p. 9]

[24] M. Garden and G. Dudek. *Mixed Collaborative and Content-based Filtering w6ith User-Contributed Semantic Features.* http://www.mcrcim.mcgill.edu/ mrl/pubs/mgarden/AAAI06GardenM.pdf, 2006. [cited at p. 18]

[25] D. Goldberg, D. Nichols, B. M. Oki, and D. Terry. *Using collaborative filtering to weave an information tapestry*, volume 35. ACM Press, http://doi.acm.org/10.1145/138859.138867, 1992. 61-70. [cited at p. 1, 10, 13]

[26] A. Gulli and A. Signorini. *The indexable web is more than 11.5 billion pages.* In Proceedings of 14th International World Wide Web Conference, Chiba, Japan, 2005. pages 902 903. [cited at p. 1]

[27] J. L. Herlocker. *Understanding and Improving Automated Collaborative Filtering Systems.* PhD Thesis, 2000. [cited at p. 7, 9, 19, 36, 37]

[28] J. L. Herlocker, L. G. Terveen J. A. Konstan, and J. T. Riedl. *Evaluating collaborative filtering recommender systems*, volume 22. ACM Transactions on information systems, 2004. [cited at p. 55, 57, 58]

[29] J. L. Herlocker, J. A. Konstan, A. Borchers, and J. Riedl. *An algorithmic framework for performing collaborative filtering.* ACM Press, http://doi.acm.org/10.1145/312624.312682, 1999. [cited at p. 13, 19, 55, 56, 58, 59]

[30] J. L. Herlocker, J. A. Konstan, and J. Riedl. *Explaining collaborative filtering recommendations.* citeseer.ist.psu.edu/herlocker00explaining.html, 2000. [cited at p. 11, 20]

[31] W. Hill, L. Stead, M. Rosenstein, and G. Furnas. *Recommending and evaluating choices in a virtual community of use.* ACM Press/Addison-Wesley Publishing Co., http://doi.acm.org/10.1145/223904.223929, 1995. 210-217. [cited at p. 12]

[32] T. Hofmann. *Latent semantic models for collaborative filtering.* ACM Trans. Inf. Syst., http://doi.acm.org/10.1145/963770.963774, 2004. [cited at p. 14]

[33] Z. Huang, H. Chen, and D. Zeng. *Applying associative retrieval techniques to alleviate the sparsity problem in collaborative filtering.* ACM Press, http://doi.acm.org/10.1145/963770.963775, 2004. [cited at p. 15]

[34] A. Jennings and H. Higuchi. *A personal news service based on a user model neural network.* citeseer.ist.psu.edu/jennings92personal.html, March 1992. [cited at p. 10]

[35] R. Jin, J. Y. Chai, and L. Si. *An automatic weighting scheme for collaborative filtering.* SIGIR '04: Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval, http://doi.acm.org/10.1145/1008992.1009051, 2004. [cited at p. 13]

[36] Y. Kodama, S. Gayama, Y. Suzuki, S. Odagawa, T. Shioda, F. Matsushita, and T. Tabata. *A Music Recommendation System.* http://ieeexplore.ieee.org/iel5/9776/30841/01429796.pdf, 2005. [cited at p. 19]

[37] J. A. Konstan, B. N. Miller, D. Maltz, J. L. Herlocker, L. R. Gordon, and J. Riedl. *GroupLens: Applying Collaborative Filtering to Usenet News*, volume 40. ACM Press, citeseer.ist.psu.edu/konstan97grouplens.html, 1997. 77-87. [cited at p. 12]

[38] Steve Krause. *Pandora and Last.fm: Nature vs. Nurure in Music Recommenders.* http://www.stevekrause.org/steve_krause_blog/2006/01/pandora_and_las.html, 2006. [cited at p. 20, 21]

[39] D. Lewis and K. Sparck-Jones. *Natural Language Processing for Information Retrieval*, volume 39. Communications of the ACM, citeseer.ist.psu.edu/fuhr91probabilistic.html, 1996. 92-101. [cited at p. 9]

[40] H. Lieberman. *Letizia: An Agent That Assists Web Browsing.* Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95), citeseer.ist.psu.edu/lieberman95letizia.html, 1995. 924-929. [cited at p. 9]

[41] G. Linden, B. Smith, and J. York. *Amazon.com Recommendations: Item-to-Item Collaborative Filtering.* IEEE Internet Computing, http://dx.doi.org/10.1109/MIC.2003.1167344, 2003. [cited at p. 14]

[42] P. Massa and P. Avesani. *Trust-aware collaborative filtering for recommender systems.* In Proc. of Federated Int. Conference On The Move to Meaningful Internet: CoopIS, DOA, ODBASE,, citeseer.ist.psu.edu/article/massa04trustaware.html, 2004. [cited at p. 10]

[43] P. Melville, R. Mooney, and R. Nagarajan. *Content-boosted collaborative filtering*, volume 40. In Proceedings of the 2001., citeseer.ist.psu.edu/melville01contentboosted.html, 2001. [cited at p. 1]

[44] H. S. Park, J. O. Yoo, and S. B. Cho. *A Context-Aware Music Recommendation System Using Fuzzy Bayesian Networks with Utility Theory.* http://sclab.yonsei.ac.kr/publications/Papers/LNCS/FSKD2006_PHS.pdf, 2006. [cited at p. 18]

[45] M. J. Pazzani, J. Muramatsu, and D. Billsus. *Syskill & Webert: Identifying Interesting Web Sites.* (AAAI)/(IAAI), Vol. 1, citeseer.ist.psu.edu/article/pazzani98syskill.html, 1996. 54-61. [cited at p. 9, 17]

[46] Michael J. Pazzani. *A Framework for Collaborative, Content-Based and Demographic Filtering.* Artificial Intelligence Review, citeseer.ist.psu.edu/pazzani99framework.html, 1999. [cited at p. 16, 20]

[47] D. Pennock, E. Horvitz, S. Lawrence, and C. L. Giles. *Collaborative Filtering by Personality Diagnosis: A Hybrid Memory- and Model-based Approach.* Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence, UAI, citeseer.ist.psu.edu/pennock00collaborative.html, 2000. [cited at p. 13, 15, 20]

[48] P. Resnick, N. Iacovou, M. Suchak, P. Bergstorm, and J. Riedl. *GroupLens: An Open Architecture for Collaborative Filtering of Netnews.* Proceedings of ACM Conference on Computer Supported Cooperative Work, citeseer.ist.psu.edu/resnick94grouplens.html, 1994. [cited at p. 1, 7, 12, 13, 14, 17, 36, 37]

[49] Paul Resnick and Hal R. Varian. *Recommender systems*, volume 40. ACM Press, http://doi.acm.org/10.1145/245108.245121, 1997. [cited at p. 1, 8]

[50] G. Salton and C. Buckley. *Information Processing and Management.* Term Weighting Approaches in Automatic Text Retrieval, 1988. 513-523. [cited at p. 9]

[51] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. Riedl. *Analysis of recommendation algorithms for e-commerce.* ACM Conference on Electronic Commerce, citeseer.ist.psu.edu/article/sarwar00analysis.html, 2000. [cited at p. 15]

[52] Badrul M. Sarwar, George Karypis, Joseph A. Konstan, and John Reidl. *Item-based collaborative filtering recommendation algorithms.* citeseer.ist.psu.edu/sarwar01itembased.html, 2001. [cited at p. 1, 14, 20]

[53] J. B. Schafer, J. A. Konstan, and J. Riedl. *E-Commerce Recommendation Applications.* Data Mining and Knowledge Discovery, citeseer.ist.psu.edu/schafer01ecommerce.html, 2001. [cited at p. 8, 9, 20]

[54] E. Selberg. *Towards Comprehensive Web Search.* PhD thesis, University of Washington, 1999. [cited at p. 1]

[55] U. Shardanand and P. Maes. *Social information filtering: algorithms for automating word of mouth.* ACM Press/Addison-Wesley Publishing Co., http://doi.acm.org/10.1145/223904.223931, 1995. 210-217. [cited at p. 12, 13, 14, 36, 37]

[56] I. Soboroff and C. Nicholas. *Combining content and collaboration in text filtering.* citeseer.ist.psu.edu/soboroff99combining.html, 1999. [cited at p. 17]

[57] ISC Domain Survey. *Number of Internet Hosts.* Internet System Consortium, http://www.isc.org/index.pl?/ops/ds/host-count-history.php, 2006. [cited at p. 6]

[58] K. Swearingen and S. Rashmi. *Interaction design for recommender systems.* In Designing Interactive Systems, citeseer.ist.psu.edu/swearingen02interaction.html, 2002. [cited at p. 8]

[59] L. Ungar and D. Foster. *Clustering Methods For Collaborative Filtering.* Proceedings of the Workshop on Recommendation Systems. AAAI Press, Menlo Park California., citeseer.ist.psu.edu/article/ungar98clustering.html, 1998. [cited at p. 15]

[60] R. van Meteren and M. van Someren. *Using Content-Based Filtering for Recommendation.* citeseer.ist.psu.edu/499652.html. [cited at p. 9]

[61] J. Wang, A. P. de Vries, and M. J. T. Reinders. *Unifying user-based and item-based collaborative filtering approaches by similarity fusion.* SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval, http://doi.acm.org/10.1145/1148170.1148257, 2006. [cited at p. 12, 15]

[62] G. R. Xue, C. Lin, Q. Yang, W. Xi, H. J. Zeng, Y. Yu, and Z. Chen. *Scalable collaborative filtering using cluster-based smoothing.* ACM Press, http://doi.acm.org/10.1145/1076034.1076056, 2005. [cited at p. 15]

Appendices

List of Figures

CD-ROM

The CD-ROM contains this document in pdf format, all source files and the test results in Excel workbooks.